QuantWorks Documentation

Release 0.21rc2

Tyler M Kontra

Dec 30, 2020

Contents

1 Introduction **3**

2 Migrating from PyAlgoTrade **5**
2.1 Changes . . . 5
2.2 Backwards Compatibility . . . 5

3 Tutorial **7**
3.1 Trading . . . 11
3.2 Optimizing . . . 14
3.3 Plotting . . . 18

4 API Documentation **21**
4.1 bar – Instrument prices . . . 21
4.2 dataseries – Basic dataseries classes . . . 23
4.3 feed – Basic feeds . . . 23
4.3.1 CSV support . . . 23
4.3.2 CSV support Example . . . 23
4.4 barfeed – Bar providers . . . 25
4.4.1 CSV . . . 25
4.4.2 Yahoo! Finance . . . 25
4.4.3 Google Finance . . . 25
4.4.4 Quandl . . . 25
4.4.5 Ninja Trader . . . 25
4.4.6 SQLite . . . 25
4.5 technical – Technical indicators . . . 25
4.5.1 Example . . . 25
4.5.2 Moving Averages . . . 26
4.5.3 Momentum Indicators . . . 26
4.5.4 Other Indicators . . . 26
4.6 broker – Order management classes . . . 26
4.6.1 Base module and classes . . . 26
4.6.2 Backtesting module and classes . . . 31
4.7 strategy – Basic strategy classes . . . 39
4.7.1 Strategy . . . 40
4.7.2 Position . . . 40
4.8 stratanalyzer – Strategy analyzers . . . 40
4.8.1 Returns . . . 40

4.8.2 Sharpe Ratio . . . 40
4.8.3 DrawDown . . . 40
4.8.4 Trades . . . 41
4.8.5 Example . . . 41
4.9 plotter – Strategy plotter . . . 44
4.10 optimizer – Parallel optimizers . . . 44
4.11 marketsession – Market sessions . . . 44

5 Tools **47**
5.1 Quandl . . . 47
5.2 BarFeed resampling . . . 47

6 Event profiler **49**
6.1 Example . . . 49

7 TA-Lib integration **53**

8 Computational Investing Part I **55**
8.1 Homework 1 . . . 55

9 Sample strategies **59**
9.1 Momentum . . . 59
9.1.1 VWAP Momentum Trade . . . 59
9.1.2 SMA Crossover . . . 61
9.1.3 Market Timing Using Moving-Average Crossovers . . . 63
9.2 Mean Reversion . . . 68
9.2.1 Ernie Chan's Gold vs. Gold Miners . . . 68
9.2.2 Bollinger Bands . . . 72
9.2.3 RSI2 . . . 75
9.3 Others . . . 79
9.3.1 Quandl integration . . . 79

10 Contributing to QuantWorks **83**
10.1 Reporting Issues . . . 83
10.2 Contributing Code . . . 83

11 Indices and tables **85**

Python Module Index **87**

Index **89**

Contents:

CHAPTER 1

Introduction

QuantWorks is an event driven algorithmic trading Python library with support for:

- Backtesting with historical data from CSV files.
- Paper trading using Bitstamp live feeds.
- Real trading on Bitstamp.

It should also make it easy to optimize a strategy using multiple computers.

QuantWorks is developed and tested using Python 2.7/3.7 and depends on:

- NumPy and SciPy (http://numpy.scipy.org/).
- pytz (http://pytz.sourceforge.net/).
- matplotlib (http://matplotlib.sourceforge.net/) for plotting support.
- ws4py (https://github.com/Lawouach/WebSocket-for-Python) for Bitstamp support.
- tornado (http://www.tornadoweb.org/en/stable/) for Bitstamp support.
- tweepy (https://github.com/tweepy/tweepy) for Twitter support.

so you need to have those installed in order to use this library.

You can install QuantWorks using pip like this:

```
pip install quantworks
```

CHAPTER 2

Migrating from PyAlgoTrade

QuantWorks is a fork of PyAlgoTrade. It should function as a drop-in replacement of `pyalgotrade` (as of v0.20), with a few minor tweaks.

Please keep in mind QuantWorks is a **Python 3** ONLY package, Python 2 support is not intended or guaranteed.

2.1 Changes

- convert all `pyalgotrade` imports to `quantworks`
- **BitcoinCharts & Bitstamp**
  - `pip install quantworks-bitcoin`
  - convert `pyalgotrade.bitstamp` imports to `quantworks.ext.bitstamp`
  - and `pyalgotrade.bitcoincharts` to `quantworks.ext.bitcoincharts`
- **Twitter**
  - `pip install quantworks-twitter`
  - convert `pyalgotrade.twitter` imports to `quantworks.ext.twitter`

If you uncover any compatibility issues with pyalgotrade please open an issue on GitHub.

2.2 Backwards Compatibility

QuantWorks was forked from PyAlgoTrade v0.20, the first release of QuantWorks being v0.21.

Backwards compatibility is not guaranteed as the public API changes until the v1.0 release.

If you feel strongly about keeping backwards compatibility in one or more specific modules, please open an issue to discuss.

CHAPTER 3

Tutorial

By Gabriel Martin Becedillas Ruiz

The goal of this tutorial is to give you a quick introduction to QuantWorks. As described in the introduction, the goal of QuantWorks is to help you backtest stock trading strategies. Let's say you have an idea for a trading strategy and you'd like to evaluate it with historical data and see how it behaves, then QuantWorks should allow you to do so with minimal effort.

Before I move on I would like to thank Pablo Jorge who helped reviewing the initial design and documentation.

This tutorial was developed on a UNIX environment, but the steps to adapt it to a Windows environment should be straightforward.

QuantWorks has 6 main components:

- Strategies
- Feeds
- Brokers
- DataSeries
- Technicals
- Optimizer

Strategies These are the classes that you define that implement the trading logic. When to buy, when to sell, etc.

Feeds These are data providing abstractions. For example, you'll use a CSV feed that loads bars from a CSV (Comma-separated values) formatted file to feed data to a strategy. Feeds are not limited to bars. For example, there is a Twitter feed that allows incorporating Twitter events into trading decisions.

Brokers Brokers are responsible for executing orders.

DataSeries A data series is an abstraction used to manage time series data.

Technicals These are a set of filters that you use to make calculations on top of DataSeries. For example SMA (Simple Moving Average), RSI (Relative Strength Index), etc. These filters are modeled as DataSeries decorators.

Optimizer These are a set of classes that allow you to distribute backtesting among different computers, or different processes running in the same computer, or a combination of both. They make horizontal scaling easy.

Having said all that, the first thing that we'll need to test our strategies is some data. Let's use Oracle's stock prices for year 2000, which we'll download with the following command:

```
python -m "quantworks.tools.quandl" --source-code="WIKI" --table-code="ORCL" --from-
↪year=2000 --to-year=2000 --storage=. --force-download --frequency=daily
```

The quantworks.tools.quandl tool downloads CSV formatted data from Quandl. The first few lines of `WIKI-ORCL-2000-quandl.csv` should look like this:

```
Date,Open,High,Low,Close,Volume,Ex-Dividend,Split Ratio,Adj. Open,Adj. High,Adj. Low,
↪Adj. Close,Adj. Volume
2000-12-29,30.88,31.31,28.69,29.06,31702200.0,0.0,1.0,28.121945213877797,28.
↪513539658242028,26.127545601883227,26.46449896098733,31702200.0
2000-12-28,30.56,31.63,30.38,31.06,25053600.0,0.0,1.0,27.830526092490462,28.
↪804958779629363,27.666602836710087,28.285868469658173,25053600.0
2000-12-27,30.38,31.06,29.38,30.69,26437500.0,0.0,1.0,27.666602836710087,28.
↪285868469658173,26.755918082374667,27.94891511055407,26437500.0
2000-12-26,31.5,32.19,30.0,30.94,20589500.0,0.0,1.0,28.68656976156576,29.
↪3149422420572,27.32054263006263,28.176586299137927,20589500.0
2000-12-22,30.38,31.98,30.0,31.88,35568200.0,0.0,1.0,27.666602836710087,29.
↪123698443646763,27.32054263006263,29.032629968213218,35568200.0
2000-12-21,27.81,30.25,27.31,29.5,46719700.0,0.0,1.0,25.326143018068056,27.
↪548213818646484,24.870800640900345,26.86520025289492,46719700.0
2000-12-20,28.06,29.81,27.5,28.5,54440500.0,0.0,1.0,25.55381420665191,27.
↪147512526738897,25.043830744224078,25.9545154985595,54440500.0
2000-12-19,31.81,33.13,30.13,30.63,58653700.0,0.0,1.0,28.968882035409738,30.
↪170985911132497,27.438931648126232,27.894274025293942,58653700.0
2000-12-18,30.0,32.44,29.94,32.0,61640100.0,0.0,1.0,27.32054263006263,29.
↪542613430641055,27.265901544802503,29.14191213873347,61640100.0
```

Let's start with a simple strategy, that is, one that just prints closing prices as they are processed:

```
from quantworks import strategy
from quantworks.barfeed import quandlfeed


class MyStrategy(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument):
        super(MyStrategy, self).__init__(feed)
        self.__instrument = instrument

    def onBars(self, bars):
        bar = bars[self.__instrument]
        self.info(bar.getClose())

# Load the bar feed from the CSV file
feed = quandlfeed.Feed()
feed.addBarsFromCSV("orcl", "WIKI-ORCL-2000-quandl.csv")

# Evaluate the strategy with the feed's bars.
myStrategy = MyStrategy(feed, "orcl")
myStrategy.run()
```

The code is doing 3 main things:

1. Declaring a new strategy. There is only one method that has to be defined, *onBars*, which is called for

every bar in the feed.

2. Loading the feed from a CSV file.
3. Running the strategy with the bars supplied by the feed.

If you run the script you should see the closing prices in order:

```
2000-01-03 00:00:00 strategy [INFO] 118.1
2000-01-04 00:00:00 strategy [INFO] 107.7
2000-01-05 00:00:00 strategy [INFO] 103.5
.
.
.
2000-12-27 00:00:00 strategy [INFO] 30.69
2000-12-28 00:00:00 strategy [INFO] 31.06
2000-12-29 00:00:00 strategy [INFO] 29.06
```

Let's move on with a strategy that prints closing SMA prices, to illustrate how technicals are used:

```
from quantworks import strategy
from quantworks.barfeed import quandlfeed
from quantworks.technical import ma


def safe_round(value, digits):
    if value is not None:
        value = round(value, digits)
    return value


class MyStrategy(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument):
        super(MyStrategy, self).__init__(feed)
        # We want a 15 period SMA over the closing prices.
        self.__sma = ma.SMA(feed[instrument].getCloseDataSeries(), 15)
        self.__instrument = instrument

    def onBars(self, bars):
        bar = bars[self.__instrument]
        self.info("%s %s" % (bar.getClose(), safe_round(self.__sma[-1], 2)))


# Load the bar feed from the CSV file
feed = quandlfeed.Feed()
feed.addBarsFromCSV("orcl", "WIKI-ORCL-2000-quandl.csv")

# Evaluate the strategy with the feed's bars.
myStrategy = MyStrategy(feed, "orcl")
myStrategy.run()
```

This is very similar to the previous example, except that:

1. We're initializing an SMA filter over the closing price data series.
2. We're printing the current SMA value along with the closing price.

If you run the script you should see the closing prices and the corresponding SMA values, but in this case the first 14 SMA values are None. That is because we need at least 15 values to get something out of the SMA:

```
2000-01-03 00:00:00 strategy [INFO] 118.1 None
2000-01-04 00:00:00 strategy [INFO] 107.7 None
2000-01-05 00:00:00 strategy [INFO] 103.5 None
2000-01-06 00:00:00 strategy [INFO] 96.0 None
2000-01-07 00:00:00 strategy [INFO] 103.4 None
2000-01-10 00:00:00 strategy [INFO] 115.8 None
2000-01-11 00:00:00 strategy [INFO] 112.4 None
2000-01-12 00:00:00 strategy [INFO] 105.6 None
2000-01-13 00:00:00 strategy [INFO] 105.1 None
2000-01-14 00:00:00 strategy [INFO] 106.8 None
2000-01-18 00:00:00 strategy [INFO] 111.3 None
2000-01-19 00:00:00 strategy [INFO] 57.13 None
2000-01-20 00:00:00 strategy [INFO] 59.25 None
2000-01-21 00:00:00 strategy [INFO] 59.69 None
2000-01-24 00:00:00 strategy [INFO] 54.19 94.4
2000-01-25 00:00:00 strategy [INFO] 56.44 90.29
.
.
.
2000-12-27 00:00:00 strategy [INFO] 30.69 29.99
2000-12-28 00:00:00 strategy [INFO] 31.06 30.05
2000-12-29 00:00:00 strategy [INFO] 29.06 30.1
```

All the technicals will return None when the value can't be calculated at a given time.

One important thing about technicals is that they can be combined. That is because they're modeled as DataSeries as well. For example, getting an SMA over the RSI over the closing prices is as simple as this:

```
from quantworks import strategy
from quantworks.barfeed import quandlfeed
from quantworks.technical import ma
from quantworks.technical import rsi


def safe_round(value, digits):
    if value is not None:
        value = round(value, digits)
    return value


class MyStrategy(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument):
        super(MyStrategy, self).__init__(feed)
        self.__rsi = rsi.RSI(feed[instrument].getCloseDataSeries(), 14)
        self.__sma = ma.SMA(self.__rsi, 15)
        self.__instrument = instrument

    def onBars(self, bars):
        bar = bars[self.__instrument]
        self.info("%s %s %s" % (
            bar.getClose(), safe_round(self.__rsi[-1], 2), safe_round(self.__sma[-1],␣
↪2)
        ))


# Load the bar feed from the CSV file
feed = quandlfeed.Feed()
feed.addBarsFromCSV("orcl", "WIKI-ORCL-2000-quandl.csv")
```

(continues on next page)

(continued from previous page)

```
# Evaluate the strategy with the feed's bars.
myStrategy = MyStrategy(feed, "orcl")
myStrategy.run()
```

If you run the script you should see a bunch of values on the screen where:

- The first 14 RSI values are None. That is because we need at least 15 values to get an RSI value.
- The first 28 SMA values are None. That is because the first 14 RSI values are None, and the 15th one is the first not None value that the SMA filter receives. We can calculate the SMA(15) only when we have 15 not None values .

```
2000-01-03 00:00:00 strategy [INFO] 118.1 None None
2000-01-04 00:00:00 strategy [INFO] 107.7 None None
2000-01-05 00:00:00 strategy [INFO] 103.5 None None
2000-01-06 00:00:00 strategy [INFO] 96.0 None None
2000-01-07 00:00:00 strategy [INFO] 103.4 None None
2000-01-10 00:00:00 strategy [INFO] 115.8 None None
2000-01-11 00:00:00 strategy [INFO] 112.4 None None
2000-01-12 00:00:00 strategy [INFO] 105.6 None None
2000-01-13 00:00:00 strategy [INFO] 105.1 None None
2000-01-14 00:00:00 strategy [INFO] 106.8 None None
2000-01-18 00:00:00 strategy [INFO] 111.3 None None
2000-01-19 00:00:00 strategy [INFO] 57.13 None None
2000-01-20 00:00:00 strategy [INFO] 59.25 None None
2000-01-21 00:00:00 strategy [INFO] 59.69 None None
2000-01-24 00:00:00 strategy [INFO] 54.19 23.6 None
2000-01-25 00:00:00 strategy [INFO] 56.44 25.1 None
2000-01-26 00:00:00 strategy [INFO] 55.06 24.78 None
2000-01-27 00:00:00 strategy [INFO] 51.81 24.0 None
2000-01-28 00:00:00 strategy [INFO] 47.38 22.94 None
2000-01-31 00:00:00 strategy [INFO] 49.95 25.01 None
2000-02-01 00:00:00 strategy [INFO] 54.0 28.27 None
2000-02-02 00:00:00 strategy [INFO] 54.31 28.53 None
2000-02-03 00:00:00 strategy [INFO] 56.69 30.58 None
2000-02-04 00:00:00 strategy [INFO] 57.81 31.58 None
2000-02-07 00:00:00 strategy [INFO] 59.94 33.53 None
2000-02-08 00:00:00 strategy [INFO] 59.56 33.35 None
2000-02-09 00:00:00 strategy [INFO] 59.94 33.73 None
2000-02-10 00:00:00 strategy [INFO] 62.31 36.23 None
2000-02-11 00:00:00 strategy [INFO] 59.69 34.68 29.06
2000-02-14 00:00:00 strategy [INFO] 62.19 37.44 29.98
.
.
.
2000-12-27 00:00:00 strategy [INFO] 30.69 51.31 49.85
2000-12-28 00:00:00 strategy [INFO] 31.06 52.16 50.0
2000-12-29 00:00:00 strategy [INFO] 29.06 47.37 50.08
```

3.1 Trading

Let's move on with a simple strategy, this time simulating actual trading. The idea is very simple:

- If the adjusted close price is above the SMA(15) we enter a long position (we place a buy market order).

- If a long position is in place, and the adjusted close price drops below the SMA(15) we exit the long position (we place a sell market order).

```
from __future__ import print_function

from quantworks import strategy
from quantworks.barfeed import quandlfeed
from quantworks.technical import ma


class MyStrategy(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument, smaPeriod):
        super(MyStrategy, self).__init__(feed, 1000)
        self.__position = None
        self.__instrument = instrument
        # We'll use adjusted close values instead of regular close values.
        self.setUseAdjustedValues(True)
        self.__sma = ma.SMA(feed[instrument].getPriceDataSeries(), smaPeriod)

    def onEnterOk(self, position):
        execInfo = position.getEntryOrder().getExecutionInfo()
        self.info("BUY at $%.2f" % (execInfo.getPrice()))

    def onEnterCanceled(self, position):
        self.__position = None

    def onExitOk(self, position):
        execInfo = position.getExitOrder().getExecutionInfo()
        self.info("SELL at $%.2f" % (execInfo.getPrice()))
        self.__position = None

    def onExitCanceled(self, position):
        # If the exit was canceled, re-submit it.
        self.__position.exitMarket()

    def onBars(self, bars):
        # Wait for enough bars to be available to calculate a SMA.
        if self.__sma[-1] is None:
            return

        bar = bars[self.__instrument]
        # If a position was not opened, check if we should enter a long position.
        if self.__position is None:
            if bar.getPrice() > self.__sma[-1]:
                # Enter a buy market order for 10 shares. The order is good till␣
↪canceled.
                self.__position = self.enterLong(self.__instrument, 10, True)
        # Check if we have to exit the position.
        elif bar.getPrice() < self.__sma[-1] and not self.__position.exitActive():
            self.__position.exitMarket()


def run_strategy(smaPeriod):
    # Load the bar feed from the CSV file
    feed = quandlfeed.Feed()
    feed.addBarsFromCSV("orcl", "WIKI-ORCL-2000-quandl.csv")

    # Evaluate the strategy with the feed.
```

(continues on next page)

(continued from previous page)

```
    myStrategy = MyStrategy(feed, "orcl", smaPeriod)
    myStrategy.run()
    print("Final portfolio value: $%.2f" % myStrategy.getBroker().getEquity())

run_strategy(15)
```

If you run the script you should see something like this:

```
2000-01-26 00:00:00 strategy [INFO] BUY at $25.84
2000-01-28 00:00:00 strategy [INFO] SELL at $23.45
2000-02-03 00:00:00 strategy [INFO] BUY at $25.22
2000-02-22 00:00:00 strategy [INFO] SELL at $26.92
2000-02-23 00:00:00 strategy [INFO] BUY at $27.41
2000-03-31 00:00:00 strategy [INFO] SELL at $36.51
2000-04-07 00:00:00 strategy [INFO] BUY at $38.11
2000-04-12 00:00:00 strategy [INFO] SELL at $35.49
2000-04-19 00:00:00 strategy [INFO] BUY at $35.80
2000-04-20 00:00:00 strategy [INFO] SELL at $33.61
2000-04-28 00:00:00 strategy [INFO] BUY at $35.74
2000-05-05 00:00:00 strategy [INFO] SELL at $33.70
2000-05-08 00:00:00 strategy [INFO] BUY at $34.29
2000-05-09 00:00:00 strategy [INFO] SELL at $33.55
2000-05-16 00:00:00 strategy [INFO] BUY at $35.35
2000-05-19 00:00:00 strategy [INFO] SELL at $32.78
2000-05-31 00:00:00 strategy [INFO] BUY at $33.35
2000-06-23 00:00:00 strategy [INFO] SELL at $36.80
2000-06-27 00:00:00 strategy [INFO] BUY at $37.51
2000-06-28 00:00:00 strategy [INFO] SELL at $37.37
2000-06-29 00:00:00 strategy [INFO] BUY at $37.37
2000-06-30 00:00:00 strategy [INFO] SELL at $36.60
2000-07-03 00:00:00 strategy [INFO] BUY at $36.94
2000-07-05 00:00:00 strategy [INFO] SELL at $34.97
2000-07-21 00:00:00 strategy [INFO] BUY at $35.26
2000-07-24 00:00:00 strategy [INFO] SELL at $35.12
2000-07-26 00:00:00 strategy [INFO] BUY at $34.06
2000-07-28 00:00:00 strategy [INFO] SELL at $34.21
2000-08-01 00:00:00 strategy [INFO] BUY at $34.24
2000-08-02 00:00:00 strategy [INFO] SELL at $33.24
2000-08-04 00:00:00 strategy [INFO] BUY at $35.66
2000-09-11 00:00:00 strategy [INFO] SELL at $39.19
2000-09-29 00:00:00 strategy [INFO] BUY at $37.05
2000-10-02 00:00:00 strategy [INFO] SELL at $36.31
2000-10-20 00:00:00 strategy [INFO] BUY at $32.90
2000-10-31 00:00:00 strategy [INFO] SELL at $29.72
2000-11-20 00:00:00 strategy [INFO] BUY at $22.14
2000-11-21 00:00:00 strategy [INFO] SELL at $22.59
2000-12-01 00:00:00 strategy [INFO] BUY at $24.02
2000-12-15 00:00:00 strategy [INFO] SELL at $26.81
2000-12-18 00:00:00 strategy [INFO] BUY at $27.32
2000-12-21 00:00:00 strategy [INFO] SELL at $25.33
2000-12-22 00:00:00 strategy [INFO] BUY at $27.67
Final portfolio value: $974.87
```

But what if we used 30 as the SMA period instead of 15 ? Would that yield better results or worse ? We could certainly do something like this:

```
for i in range(10, 30):
    run_strategy(i)
```

and we would find out that we can get better results with a SMA(20):

```
Final portfolio value: $1071.03
```

This is ok if we only have to try a limited set of parameters values. But if we have to test a strategy with multiple parameters, then the serial approach is definitely not going to scale as strategies get more complex.

3.2 Optimizing

Meet the optimizer component. The idea is very simple:

- **There is one server responsible for:**
  - Providing the bars to run the strategy.
  - Providing the parameters to run the strategy.
  - Recording the strategy results from each of the workers.
- **There are multiple workers responsible for:**
  - Running the strategy with the bars and parameters provided by the server.

To illustrate this we'll use a strategy known as RSI2 which requires the following parameters:

- An SMA period for trend identification. We'll call this entrySMA and will range between 150 and 250.
- A smaller SMA period for the exit point. We'll call this exitSMA and will range between 5 and 15.
- An RSI period for entering both short/long positions. We'll call this rsiPeriod and will range between 2 and 10.
- An RSI oversold threshold for long position entry. We'll call this overSoldThreshold and will range between 5 and 25.
- An RSI overbought threshold for short position entry. We'll call this overBoughtThreshold and will range between 75 and 95.

If my math is ok, those are 4409559 different combinations.

Testing this strategy for one set of parameters took me about 0.16 seconds. If I execute all the combinations serially it'll take me about 8.5 days to evaluate all of them and find the best set of parameters. That is a long time, but if I can get ten 8-core computers to do the job then the total time will go down to about 2.5 hours.

Long story short, **we need to go parallel**.

Let's start by downloading 3 years of daily bars for 'IBM':

```
python -m "quantworks.tools.quandl" --source-code="WIKI" --table-code="IBM" --from-
↪year=2009 --to-year=2011 --storage=. --force-download --frequency=daily
```

Save this code as rsi2.py:

```
from quantworks import strategy
from quantworks.technical import ma
from quantworks.technical import rsi
from quantworks.technical import cross
```

(continues on next page)

(continued from previous page)

```
class RSI2(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument, entrySMA, exitSMA, rsiPeriod,␣
↪overBoughtThreshold, overSoldThreshold):
        super(RSI2, self).__init__(feed)
        self.__instrument = instrument
        # We'll use adjusted close values, if available, instead of regular close␣
↪values.
        if feed.barsHaveAdjClose():
            self.setUseAdjustedValues(True)
        self.__priceDS = feed[instrument].getPriceDataSeries()
        self.__entrySMA = ma.SMA(self.__priceDS, entrySMA)
        self.__exitSMA = ma.SMA(self.__priceDS, exitSMA)
        self.__rsi = rsi.RSI(self.__priceDS, rsiPeriod)
        self.__overBoughtThreshold = overBoughtThreshold
        self.__overSoldThreshold = overSoldThreshold
        self.__longPos = None
        self.__shortPos = None

    def getEntrySMA(self):
        return self.__entrySMA

    def getExitSMA(self):
        return self.__exitSMA

    def getRSI(self):
        return self.__rsi

    def onEnterCanceled(self, position):
        if self.__longPos == position:
            self.__longPos = None
        elif self.__shortPos == position:
            self.__shortPos = None
        else:
            assert(False)

    def onExitOk(self, position):
        if self.__longPos == position:
            self.__longPos = None
        elif self.__shortPos == position:
            self.__shortPos = None
        else:
            assert(False)

    def onExitCanceled(self, position):
        # If the exit was canceled, re-submit it.
        position.exitMarket()

    def onBars(self, bars):
        # Wait for enough bars to be available to calculate SMA and RSI.
        if self.__exitSMA[-1] is None or self.__entrySMA[-1] is None or self.__rsi[-
↪1] is None:
            return

        bar = bars[self.__instrument]
        if self.__longPos is not None:
            if self.exitLongSignal():
```

(continues on next page)

(continued from previous page)

```
                self.__longPos.exitMarket()
        elif self.__shortPos is not None:
            if self.exitShortSignal():
                self.__shortPos.exitMarket()
        else:
            if self.enterLongSignal(bar):
                shares = int(self.getBroker().getCash() * 0.9 / bars[self.__
↪instrument].getPrice())
                self.__longPos = self.enterLong(self.__instrument, shares, True)
            elif self.enterShortSignal(bar):
                shares = int(self.getBroker().getCash() * 0.9 / bars[self.__
↪instrument].getPrice())
                self.__shortPos = self.enterShort(self.__instrument, shares, True)

    def enterLongSignal(self, bar):
        return bar.getPrice() > self.__entrySMA[-1] and self.__rsi[-1] <= self.__
↪overSoldThreshold

    def exitLongSignal(self):
        return cross.cross_above(self.__priceDS, self.__exitSMA) and not self.__
↪longPos.exitActive()

    def enterShortSignal(self, bar):
        return bar.getPrice() < self.__entrySMA[-1] and self.__rsi[-1] >= self.__
↪overBoughtThreshold

    def exitShortSignal(self):
        return cross.cross_below(self.__priceDS, self.__exitSMA) and not self.__
↪shortPos.exitActive()
```

This is the server script:

```
import itertools
from quantworks.optimizer import server
from quantworks.barfeed import quandlfeed


def parameters_generator():
    instrument = ["ibm"]
    entrySMA = range(150, 251)
    exitSMA = range(5, 16)
    rsiPeriod = range(2, 11)
    overBoughtThreshold = range(75, 96)
    overSoldThreshold = range(5, 26)
    return itertools.product(instrument, entrySMA, exitSMA, rsiPeriod,␣
↪overBoughtThreshold, overSoldThreshold)


# The if __name__ == '__main__' part is necessary if running on Windows.
if __name__ == '__main__':
    # Load the bar feed from the CSV files.
    feed = quandlfeed.Feed()
    feed.addBarsFromCSV("ibm", "WIKI-IBM-2009-quandl.csv")
    feed.addBarsFromCSV("ibm", "WIKI-IBM-2010-quandl.csv")
    feed.addBarsFromCSV("ibm", "WIKI-IBM-2011-quandl.csv")

    # Run the server.
```

(continues on next page)

(continued from previous page)

```
    server.serve(feed, parameters_generator(), "localhost", 5000)
```

The server code is doing 3 things:

1. Declaring a generator function that yields different parameter combinations for the strategy.
2. Loading the feed with the CSV files we downloaded.
3. Running the server that will wait for incoming connections on port 5000.

This is the worker script that uses the **quantworks.optimizer.worker** module to run the strategy in parallel with the data supplied by the server:

```
from quantworks.optimizer import worker
import rsi2

# The if __name__ == '__main__' part is necessary if running on Windows.
if __name__ == '__main__':
    worker.run(rsi2.RSI2, "localhost", 5000, workerName="localworker")
```

When you run the server and the client/s you'll see something like this on the server console:

```
2017-07-21 22:56:51,944 quantworks.optimizer.server [INFO] Starting server
2017-07-21 22:56:51,944 quantworks.optimizer.xmlrpcserver [INFO] Loading bars
2017-07-21 22:56:52,609 quantworks.optimizer.xmlrpcserver [INFO] Started serving
2017-07-21 22:58:50,073 quantworks.optimizer.xmlrpcserver [INFO] Best result so far␣
↪1261295.07089 with parameters ('ibm', 150, 5, 2, 83, 24)
.
.
```

and something like this on the worker/s console:

```
2017-07-21 22:56:57,884 localworker [INFO] Started running
2017-07-21 22:56:57,884 localworker [INFO] Started running
2017-07-21 22:56:58,439 localworker [INFO] Running strategy with parameters ('ibm',␣
↪150, 5, 2, 84, 15)
2017-07-21 22:56:58,498 localworker [INFO] Running strategy with parameters ('ibm',␣
↪150, 5, 2, 94, 5)
2017-07-21 22:56:58,918 localworker [INFO] Result 1137855.88871
2017-07-21 22:56:58,918 localworker [INFO] Running strategy with parameters ('ibm',␣
↪150, 5, 2, 84, 14)
2017-07-21 22:56:58,996 localworker [INFO] Result 1027761.85581
2017-07-21 22:56:58,997 localworker [INFO] Running strategy with parameters ('ibm',␣
↪150, 5, 2, 93, 25)
2017-07-21 22:56:59,427 localworker [INFO] Result 1092194.67448
2017-07-21 22:57:00,016 localworker [INFO] Result 1260766.64479
.
.
```

Note that you should run **only one server and one or more workers**.

If you just want to run strategies in parallel in your own desktop you can take advantage of the **quantworks.optimizer.local** module like this:

```
import itertools
from quantworks.optimizer import local
from quantworks.barfeed import quandlfeed
import rsi2
```

(continues on next page)

(continued from previous page)

```
def parameters_generator():
    instrument = ["ibm"]
    entrySMA = range(150, 251)
    exitSMA = range(5, 16)
    rsiPeriod = range(2, 11)
    overBoughtThreshold = range(75, 96)
    overSoldThreshold = range(5, 26)
    return itertools.product(instrument, entrySMA, exitSMA, rsiPeriod,␣
↪overBoughtThreshold, overSoldThreshold)


# The if __name__ == '__main__' part is necessary if running on Windows.
if __name__ == '__main__':
    # Load the bar feed from the CSV files.
    feed = quandlfeed.Feed()
    feed.addBarsFromCSV("ibm", "WIKI-IBM-2009-quandl.csv")
    feed.addBarsFromCSV("ibm", "WIKI-IBM-2010-quandl.csv")
    feed.addBarsFromCSV("ibm", "WIKI-IBM-2011-quandl.csv")

    local.run(rsi2.RSI2, feed, parameters_generator())
```

The code is doing 3 things:

1. Declaring a generator function that yields different parameter combinations.
2. Loading the feed with the CSV files we downloaded.
3. Using the **quantworks.optimizer.local** module to run the strategy in parallel and find the best result.

When you run this code you should see something like this:

```
2017-07-21 22:59:26,921 quantworks.optimizer.local [INFO] Starting server
2017-07-21 22:59:26,922 quantworks.optimizer.xmlrpcserver [INFO] Loading bars
2017-07-21 22:59:26,922 quantworks.optimizer.local [INFO] Starting workers
2017-07-21 22:59:27,642 quantworks.optimizer.xmlrpcserver [INFO] Started serving
2017-07-21 23:01:14,306 quantworks.optimizer.xmlrpcserver [INFO] Best result so far␣
↪1261295.07089 with parameters ('ibm', 150, 5, 2, 83, 24)
.
.
```

3.3 Plotting

QuantWorks makes it very easy to plot a strategy execution.

Save this as sma_crossover.py:

```
from quantworks import strategy
from quantworks.technical import ma
from quantworks.technical import cross


class SMACrossOver(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument, smaPeriod):
        super(SMACrossOver, self).__init__(feed)
```

(continues on next page)

(continued from previous page)

```
        self.__instrument = instrument
        self.__position = None
        # We'll use adjusted close values instead of regular close values.
        self.setUseAdjustedValues(True)
        self.__prices = feed[instrument].getPriceDataSeries()
        self.__sma = ma.SMA(self.__prices, smaPeriod)

    def getSMA(self):
        return self.__sma

    def onEnterCanceled(self, position):
        self.__position = None

    def onExitOk(self, position):
        self.__position = None

    def onExitCanceled(self, position):
        # If the exit was canceled, re-submit it.
        self.__position.exitMarket()

    def onBars(self, bars):
        # If a position was not opened, check if we should enter a long position.
        if self.__position is None:
            if cross.cross_above(self.__prices, self.__sma) > 0:
                shares = int(self.getBroker().getCash() * 0.9 / bars[self.__
↪instrument].getPrice())
                # Enter a buy market order. The order is good till canceled.
                self.__position = self.enterLong(self.__instrument, shares, True)
        # Check if we have to exit the position.
        elif not self.__position.exitActive() and cross.cross_below(self.__prices,␣
↪self.__sma) > 0:
            self.__position.exitMarket()
```

and save this code to a different file:

```
from quantworks import plotter
from quantworks.barfeed import quandlfeed
from quantworks.stratanalyzer import returns
import sma_crossover

# Load the bar feed from the CSV file
feed = quandlfeed.Feed()
feed.addBarsFromCSV("orcl", "WIKI-ORCL-2000-quandl.csv")

# Evaluate the strategy with the feed's bars.
myStrategy = sma_crossover.SMACrossOver(feed, "orcl", 20)

# Attach a returns analyzers to the strategy.
returnsAnalyzer = returns.Returns()
myStrategy.attachAnalyzer(returnsAnalyzer)

# Attach the plotter to the strategy.
plt = plotter.StrategyPlotter(myStrategy)
# Include the SMA in the instrument's subplot to get it displayed along with the␣
↪closing prices.
plt.getInstrumentSubplot("orcl").addDataSeries("SMA", myStrategy.getSMA())
# Plot the simple returns on each bar.
```

(continues on next page)

(continued from previous page)

```
plt.getOrCreateSubplot("returns").addDataSeries("Simple returns", returnsAnalyzer.
↪getReturns())

# Run the strategy.
myStrategy.run()
myStrategy.info("Final portfolio value: $%.2f" % myStrategy.getResult())

# Plot the strategy.
plt.plot()
```

The code is doing 3 things:

1. Loading the feed from a CSV file.
2. Running the strategy with the bars supplied by the feed and a StrategyPlotter attached.
3. Plotting the strategy.

This is what the plot looks like:

I hope you enjoyed this quick introduction. I'd recommend you to download QuantWorks here: http://gbeced.github.io/quantworks/downloads/index.html and get started writing you own strategies.

You can also find more examples in the *Sample strategies* section.

CHAPTER 4

API Documentation

Contents:

4.1 bar – Instrument prices

class quantworks.bar.**Frequency**
Bases: object

Enum like class for bar frequencies. Valid values are:

- **Frequency.TRADE**: The bar represents a single trade.
- **Frequency.SECOND**: The bar summarizes the trading activity during 1 second.
- **Frequency.MINUTE**: The bar summarizes the trading activity during 1 minute.
- **Frequency.HOUR**: The bar summarizes the trading activity during 1 hour.
- **Frequency.DAY**: The bar summarizes the trading activity during 1 day.
- **Frequency.WEEK**: The bar summarizes the trading activity during 1 week.
- **Frequency.MONTH**: The bar summarizes the trading activity during 1 month.

class quantworks.bar.**Bar**
Bases: object

A Bar is a summary of the trading activity for a security in a given period.

Note: This is a base class and should not be used directly.

getDateTime()
Returns the datetime.datetime.

getOpen(*adjusted=False*)
Returns the opening price.

getHigh(*adjusted=False*)
Returns the highest price.

getLow(*adjusted=False*)
Returns the lowest price.

getClose(*adjusted=False*)
Returns the closing price.

getVolume()
Returns the volume.

getAdjClose()
Returns the adjusted closing price.

getFrequency()
The bar's period.

getTypicalPrice()
Returns the typical price.

getPrice()
Returns the closing or adjusted closing price.

class quantworks.bar.**BasicBar**(*dateTime*, *open_*, *high*, *low*, *close*, *volume*, *adjClose*, *frequency*, *extra={}*)
Bases: *quantworks.bar.Bar*

Common bar class of OCHLV wih frequency and adjusted close if available. Extra bar values are stored as a group under BasicBar.getExtraColumns()

__init__(*dateTime*, *open_*, *high*, *low*, *close*, *volume*, *adjClose*, *frequency*, *extra={}*)
Initialize self. See help(type(self)) for accurate signature.

getDateTime()
Returns the `datetime.datetime`.

getOpen(*adjusted=False*)
Returns the opening price.

getHigh(*adjusted=False*)
Returns the highest price.

getLow(*adjusted=False*)
Returns the lowest price.

getClose(*adjusted=False*)
Returns the closing price.

getVolume()
Returns the volume.

getAdjClose()
Returns the adjusted closing price.

getFrequency()
The bar's period.

getPrice()
Returns the closing or adjusted closing price.

class quantworks.bar.**Bars**(*barDict*)
Bases: `object`

A point-in-time mapping of instrument -> *Bar*.

Parameters **barDict** (*map.*) – A map of instrument to *Bar* objects.

Note: All bars must have the same datetime.

__init__(*barDict*)
Initialize self. See help(type(self)) for accurate signature.

__getitem__(*instrument*)
Returns the *quantworks.bar.Bar* for the given instrument. If the instrument is not found an exception is raised.

__contains__(*instrument*)
Returns True if a *quantworks.bar.Bar* for the given instrument is available.

getInstruments()
Returns the instrument symbols.

getDateTime()
Returns the `datetime.datetime` for this set of bars.

getBar(*instrument*)
Returns the *quantworks.bar.Bar* for the given instrument or None if the instrument is not found.

4.2 dataseries – Basic dataseries classes

Data series are abstractions used to manage time-series data.

4.3 feed – Basic feeds

Feeds are time series data providing abstractions. When these are included in the event dispatch loop, they emit an event as new data is available. Feeds are also responsible for updating the `quantworks.dataseries.DataSeries` associated with each piece of data that the feed provides.

This package has basic feeds. For bar feeds refer to the *barfeed – Bar providers* section.

4.3.1 CSV support

4.3.2 CSV support Example

A file with the following format

```
Date,USD,GBP,EUR
2013-09-29,1333.0,831.203,986.75
2013-09-22,1349.25,842.755,997.671
2013-09-15,1318.5,831.546,993.969
2013-09-08,1387.0,886.885,1052.911
.
.
.
```

can be loaded like this:

```
from __future__ import print_function

from quantworks.feed import csvfeed

feed = csvfeed.Feed("Date", "%Y-%m-%d")
feed.addValuesFromCSV("quandl_gold_2.csv")
for dateTime, value in feed:
    print(dateTime, value)
```

and the output should look like this:

```
1968-04-07 00:00:00 {'USD': 37.0, 'GBP': 15.3875, 'EUR': ''}
1968-04-14 00:00:00 {'USD': 38.0, 'GBP': 15.8208, 'EUR': ''}
1968-04-21 00:00:00 {'USD': 37.65, 'GBP': 15.6833, 'EUR': ''}
1968-04-28 00:00:00 {'USD': 38.65, 'GBP': 16.1271, 'EUR': ''}
1968-05-05 00:00:00 {'USD': 39.1, 'GBP': 16.3188, 'EUR': ''}
1968-05-12 00:00:00 {'USD': 39.6, 'GBP': 16.5625, 'EUR': ''}
1968-05-19 00:00:00 {'USD': 41.5, 'GBP': 17.3958, 'EUR': ''}
1968-05-26 00:00:00 {'USD': 41.75, 'GBP': 17.5104, 'EUR': ''}
1968-06-02 00:00:00 {'USD': 41.95, 'GBP': 17.6, 'EUR': ''}
1968-06-09 00:00:00 {'USD': 41.25, 'GBP': 17.3042, 'EUR': ''}
.
.
.
2013-07-28 00:00:00 {'USD': 1331.0, 'GBP': 864.23, 'EUR': 1001.505}
2013-08-04 00:00:00 {'USD': 1309.25, 'GBP': 858.637, 'EUR': 986.921}
2013-08-11 00:00:00 {'USD': 1309.0, 'GBP': 843.156, 'EUR': 979.79}
2013-08-18 00:00:00 {'USD': 1369.25, 'GBP': 875.424, 'EUR': 1024.964}
2013-08-25 00:00:00 {'USD': 1377.5, 'GBP': 885.738, 'EUR': 1030.6}
2013-09-01 00:00:00 {'USD': 1394.75, 'GBP': 901.292, 'EUR': 1055.749}
2013-09-08 00:00:00 {'USD': 1387.0, 'GBP': 886.885, 'EUR': 1052.911}
2013-09-15 00:00:00 {'USD': 1318.5, 'GBP': 831.546, 'EUR': 993.969}
2013-09-22 00:00:00 {'USD': 1349.25, 'GBP': 842.755, 'EUR': 997.671}
2013-09-29 00:00:00 {'USD': 1333.0, 'GBP': 831.203, 'EUR': 986.75}
```

4.4 barfeed – Bar providers

4.4.1 CSV

4.4.2 Yahoo! Finance

4.4.3 Google Finance

4.4.4 Quandl

4.4.5 Ninja Trader

4.4.6 SQLite

4.5 technical – Technical indicators

4.5.1 Example

The following example shows how to combine an `EventWindow` and an `EventBasedFilter` to build a custom filter:

```
from __future__ import print_function

from quantworks import dataseries
from quantworks import technical


# An EventWindow is responsible for making calculations using a window of values.
class Accumulator(technical.EventWindow):
    def getValue(self):
        ret = None
        if self.windowFull():
            ret = self.getValues().sum()
        return ret

# Build a sequence based DataSeries.
seqDS = dataseries.SequenceDataSeries()
# Wrap it with a filter that will get fed as new values get added to the underlying␣
↪DataSeries.
accum = technical.EventBasedFilter(seqDS, Accumulator(3))

# Put in some values.
for i in range(0, 50):
    seqDS.append(i)

# Get some values.
print(accum[0])  # Not enough values yet.
print(accum[1])  # Not enough values yet.
print(accum[2])  # Ok, now we should have at least 3 values.
print(accum[3])

# Get the last value, which should be equal to 49 + 48 + 47.
print(accum[-1])
```

The output should be:

```
None
None
3.0
6.0
144.0
```

4.5.2 Moving Averages

4.5.3 Momentum Indicators

4.5.4 Other Indicators

4.6 broker – Order management classes

4.6.1 Base module and classes

class quantworks.broker.**Order**(*type_*, *action*, *instrument*, *quantity*, *instrumentTraits*)

Bases: object

Base class for orders.

Parameters

- **type** (Order.Type) – The order type
- **action** (Order.Action) – The order action.
- **instrument** (*string.*) – Instrument identifier.
- **quantity** (*int/float.*) – Order quantity.

Note: This is a base class and should not be used directly.

Valid **type** parameter values are:

- Order.Type.MARKET
- Order.Type.LIMIT
- Order.Type.STOP
- Order.Type.STOP_LIMIT

Valid **action** parameter values are:

- Order.Action.BUY
- Order.Action.BUY_TO_COVER
- Order.Action.SELL
- Order.Action.SELL_SHORT

---

getId()

Returns the order id.

Note: This will be None if the order was not submitted.

getType()
Returns the order type. Valid order types are:

- Order.Type.MARKET
- Order.Type.LIMIT
- Order.Type.STOP
- Order.Type.STOP_LIMIT

getSubmitDateTime()
Returns the datetime when the order was submitted.

getAction()
Returns the order action. Valid order actions are:

- Order.Action.BUY
- Order.Action.BUY_TO_COVER
- Order.Action.SELL
- Order.Action.SELL_SHORT

getState()
Returns the order state. Valid order states are:

- Order.State.INITIAL (the initial state).
- Order.State.SUBMITTED
- Order.State.ACCEPTED
- Order.State.CANCELED
- Order.State.PARTIALLY_FILLED
- Order.State.FILLED

isActive()
Returns True if the order is active.

isInitial()
Returns True if the order state is Order.State.INITIAL.

isSubmitted()
Returns True if the order state is Order.State.SUBMITTED.

isAccepted()
Returns True if the order state is Order.State.ACCEPTED.

isCanceled()
Returns True if the order state is Order.State.CANCELED.

isPartiallyFilled()
Returns True if the order state is Order.State.PARTIALLY_FILLED.

isFilled()
Returns True if the order state is Order.State.FILLED.

getInstrument()
Returns the instrument identifier.

getQuantity()
Returns the quantity.

getFilled()
Returns the number of shares that have been executed.

getRemaining()
Returns the number of shares still outstanding.

getAvgFillPrice()
Returns the average price of the shares that have been executed, or None if nothing has been filled.

getGoodTillCanceled()
Returns True if the order is good till canceled.

setGoodTillCanceled(*goodTillCanceled*)
Sets if the order should be good till canceled. Orders that are not filled by the time the session closes will be will be automatically canceled if they were not set as good till canceled

Parameters **goodTillCanceled** (*boolean.*) – True if the order should be good till canceled.

Note: This can't be changed once the order is submitted.

getAllOrNone()
Returns True if the order should be completely filled or else canceled.

setAllOrNone(*allOrNone*)
Sets the All-Or-None property for this order.

Parameters **allOrNone** (*boolean.*) – True if the order should be completely filled.

Note: This can't be changed once the order is submitted.

getExecutionInfo()
Returns the last execution information for this order, or None if nothing has been filled so far. This will be different every time an order, or part of it, gets filled.

Return type *OrderExecutionInfo*.

class quantworks.broker.**MarketOrder**(*action*, *instrument*, *quantity*, *onClose*, *instrumentTraits*)
Bases: *quantworks.broker.Order*

Base class for market orders.

Note: This is a base class and should not be used directly.

getFillOnClose()
Returns True if the order should be filled as close to the closing price as possible (Market-On-Close order).

class quantworks.broker.**LimitOrder**(*action*, *instrument*, *limitPrice*, *quantity*, *instrumentTraits*)
Bases: *quantworks.broker.Order*

Base class for limit orders.

Note: This is a base class and should not be used directly.

getLimitPrice()
Returns the limit price.

class quantworks.broker.**StopOrder**(*action*, *instrument*, *stopPrice*, *quantity*, *instrumentTraits*)
Bases: *quantworks.broker.Order*

Base class for stop orders.

Note: This is a base class and should not be used directly.

---

getStopPrice()
Returns the stop price.

class quantworks.broker.**StopLimitOrder**(*action*, *instrument*, *stopPrice*, *limitPrice*, *quantity*, *instrumentTraits*)
Bases: *quantworks.broker.Order*

Base class for stop limit orders.

Note: This is a base class and should not be used directly.

---

getStopPrice()
Returns the stop price.

getLimitPrice()
Returns the limit price.

class quantworks.broker.**OrderExecutionInfo**(*price*, *quantity*, *commission*, *dateTime*)
Bases: object

Execution information for an order.

getPrice()
Returns the fill price.

getQuantity()
Returns the quantity.

getCommission()
Returns the commission applied.

getDateTime()
Returns the datatime.datetime when the order was executed.

class quantworks.broker.**Broker**
Bases: quantworks.observer.Subject

Base class for brokers.

Note: This is a base class and should not be used directly.

---

getCash(*includeShort=True*)
Returns the available cash.

Parameters **includeShort** (*boolean.*) – Include cash from short positions.

getShares(*instrument*)
Returns the number of shares for an instrument.

getPositions()
Returns a dictionary that maps instruments to shares.

getActiveOrders(*instrument=None*)
Returns a sequence with the orders that are still active.

Parameters **instrument** (*string.*) – An optional instrument identifier to return only the active orders for the given instrument.

submitOrder(*order*)
Submits an order.

Parameters **order** (*Order.*) – The order to submit.

Note:

- After this call the order is in SUBMITTED state and an event is not triggered for this transition.
- Calling this twice on the same order will raise an exception.

---

createMarketOrder(*action*, *instrument*, *quantity*, *onClose=False*)
Creates a Market order. A market order is an order to buy or sell a stock at the best available price. Generally, this type of order will be executed immediately. However, the price at which a market order will be executed is not guaranteed.

Parameters

- **action** (*Order.Action.BUY, or Order.Action.BUY_TO_COVER, or Order.Action.SELL or Order.Action.SELL_SHORT.*) – The order action.
- **instrument** (*string.*) – Instrument identifier.
- **quantity** (*int/float.*) – Order quantity.
- **onClose** (*boolean.*) – True if the order should be filled as close to the closing price as possible (Market-On-Close order). Default is False.

Return type A *MarketOrder* subclass.

createLimitOrder(*action*, *instrument*, *limitPrice*, *quantity*)
Creates a Limit order. A limit order is an order to buy or sell a stock at a specific price or better. A buy limit order can only be executed at the limit price or lower, and a sell limit order can only be executed at the limit price or higher.

Parameters

- **action** (*Order.Action.BUY, or Order.Action.BUY_TO_COVER, or Order.Action.SELL or Order.Action.SELL_SHORT.*) – The order action.
- **instrument** (*string.*) – Instrument identifier.
- **limitPrice** (*float*) – The order price.
- **quantity** (*int/float.*) – Order quantity.

Return type A *LimitOrder* subclass.

createStopOrder(*action*, *instrument*, *stopPrice*, *quantity*)
Creates a Stop order. A stop order, also referred to as a stop-loss order, is an order to buy or sell a stock once the price of the stock reaches a specified price, known as the stop price. When the stop price is reached, a stop order becomes a market order. A buy stop order is entered at a stop price above the current market price. Investors generally use a buy stop order to limit a loss or to protect a profit on a stock that

they have sold short. A sell stop order is entered at a stop price below the current market price. Investors generally use a sell stop order to limit a loss or to protect a profit on a stock that they own.

Parameters

- **action** (*Order.Action.BUY, or Order.Action.BUY_TO_COVER, or Order.Action.SELL or Order.Action.SELL_SHORT.*) – The order action.
- **instrument** (*string.*) – Instrument identifier.
- **stopPrice** (*float*) – The trigger price.
- **quantity** (*int/float.*) – Order quantity.

Return type A *StopOrder* subclass.

createStopLimitOrder(*action*, *instrument*, *stopPrice*, *limitPrice*, *quantity*)

Creates a Stop-Limit order. A stop-limit order is an order to buy or sell a stock that combines the features of a stop order and a limit order. Once the stop price is reached, a stop-limit order becomes a limit order that will be executed at a specified price (or better). The benefit of a stop-limit order is that the investor can control the price at which the order can be executed.

Parameters

- **action** (*Order.Action.BUY, or Order.Action.BUY_TO_COVER, or Order.Action.SELL or Order.Action.SELL_SHORT.*) – The order action.
- **instrument** (*string.*) – Instrument identifier.
- **stopPrice** (*float*) – The trigger price.
- **limitPrice** (*float*) – The price for the limit order.
- **quantity** (*int/float.*) – Order quantity.

Return type A *StopLimitOrder* subclass.

cancelOrder(*order*)

Requests an order to be canceled. If the order is filled an Exception is raised.

Parameters **order** (*Order*.) – The order to cancel.

4.6.2 Backtesting module and classes

class quantworks.broker.backtesting.**Commission**

Bases: object

Base class for implementing different commission schemes.

Note: This is a base class and should not be used directly.

calculate(*order*, *price*, *quantity*)

Calculates the commission for an order execution.

Parameters

- **order** (*quantworks.broker.Order*.) – The order being executed.
- **price** (*float.*) – The price for each share.
- **quantity** (*float.*) – The order size.

Return type float.

class quantworks.broker.backtesting.**NoCommission**
Bases: *quantworks.broker.backtesting.Commission*

A *Commission* class that always returns 0.

calculate(*order*, *price*, *quantity*)
Calculates the commission for an order execution.

Parameters

- **order** (*quantworks.broker.Order*.) – The order being executed.
- **price** (*float.*) – The price for each share.
- **quantity** (*float.*) – The order size.

Return type float.

class quantworks.broker.backtesting.**FixedPerTrade**(*amount*)
Bases: *quantworks.broker.backtesting.Commission*

A *Commission* class that charges a fixed amount for the whole trade.

Parameters **amount** (*float.*) – The commission for an order.

calculate(*order*, *price*, *quantity*)
Calculates the commission for an order execution.

Parameters

- **order** (*quantworks.broker.Order*.) – The order being executed.
- **price** (*float.*) – The price for each share.
- **quantity** (*float.*) – The order size.

Return type float.

class quantworks.broker.backtesting.**TradePercentage**(*percentage*)
Bases: *quantworks.broker.backtesting.Commission*

A *Commission* class that charges a percentage of the whole trade.

Parameters **percentage** (*float.*) – The percentage to charge. 0.01 means 1%, and so on. It must be smaller than 1.

calculate(*order*, *price*, *quantity*)
Calculates the commission for an order execution.

Parameters

- **order** (*quantworks.broker.Order*.) – The order being executed.
- **price** (*float.*) – The price for each share.
- **quantity** (*float.*) – The order size.

Return type float.

class quantworks.broker.backtesting.**Broker**(*cash*, *barFeed*, *commission=None*)
Bases: *quantworks.broker.Broker*

Backtesting broker.

Parameters

- **cash** (*int/float.*) – The initial amount of cash.

- **barFeed** (quantworks.barfeed.BarFeed) – The bar feed that will provide the bars.
- **commission** (*Commission*) – An object responsible for calculating order commissions.

cancelOrder(*order*)
Requests an order to be canceled. If the order is filled an Exception is raised.

Parameters **order** (Order.) – The order to cancel.

createLimitOrder(*action*, *instrument*, *limitPrice*, *quantity*)
Creates a Limit order. A limit order is an order to buy or sell a stock at a specific price or better. A buy limit order can only be executed at the limit price or lower, and a sell limit order can only be executed at the limit price or higher.

Parameters

- **action** (*Order.Action.BUY, or Order.Action.BUY_TO_COVER, or Order.Action.SELL or Order.Action.SELL_SHORT.*) – The order action.
- **instrument** (*string.*) – Instrument identifier.
- **limitPrice** (*float*) – The order price.
- **quantity** (*int/float.*) – Order quantity.

Return type A LimitOrder subclass.

createMarketOrder(*action*, *instrument*, *quantity*, *onClose=False*)
Creates a Market order. A market order is an order to buy or sell a stock at the best available price. Generally, this type of order will be executed immediately. However, the price at which a market order will be executed is not guaranteed.

Parameters

- **action** (*Order.Action.BUY, or Order.Action.BUY_TO_COVER, or Order.Action.SELL or Order.Action.SELL_SHORT.*) – The order action.
- **instrument** (*string.*) – Instrument identifier.
- **quantity** (*int/float.*) – Order quantity.
- **onClose** (*boolean.*) – True if the order should be filled as close to the closing price as possible (Market-On-Close order). Default is False.

Return type A MarketOrder subclass.

createStopLimitOrder(*action*, *instrument*, *stopPrice*, *limitPrice*, *quantity*)
Creates a Stop-Limit order. A stop-limit order is an order to buy or sell a stock that combines the features of a stop order and a limit order. Once the stop price is reached, a stop-limit order becomes a limit order that will be executed at a specified price (or better). The benefit of a stop-limit order is that the investor can control the price at which the order can be executed.

Parameters

- **action** (*Order.Action.BUY, or Order.Action.BUY_TO_COVER, or Order.Action.SELL or Order.Action.SELL_SHORT.*) – The order action.
- **instrument** (*string.*) – Instrument identifier.
- **stopPrice** (*float*) – The trigger price.
- **limitPrice** (*float*) – The price for the limit order.
- **quantity** (*int/float.*) – Order quantity.

Return type A `StopLimitOrder` subclass.

createStopOrder(*action*, *instrument*, *stopPrice*, *quantity*)
Creates a Stop order. A stop order, also referred to as a stop-loss order, is an order to buy or sell a stock once the price of the stock reaches a specified price, known as the stop price. When the stop price is reached, a stop order becomes a market order. A buy stop order is entered at a stop price above the current market price. Investors generally use a buy stop order to limit a loss or to protect a profit on a stock that they have sold short. A sell stop order is entered at a stop price below the current market price. Investors generally use a sell stop order to limit a loss or to protect a profit on a stock that they own.

Parameters

- **action** (*Order.Action.BUY, or Order.Action.BUY_TO_COVER, or Order.Action.SELL or Order.Action.SELL_SHORT.*) – The order action.
- **instrument** (*string.*) – Instrument identifier.
- **stopPrice** (*float*) – The trigger price.
- **quantity** (*int/float.*) – Order quantity.

Return type A `StopOrder` subclass.

dispatch()
Dispatch events. If True is returned, it means that at least one event was dispatched.

eof()
Return True if there are not more events to dispatch.

getActiveOrders(*instrument=None*)
Returns a sequence with the orders that are still active.

Parameters **instrument** (*string.*) – An optional instrument identifier to return only the active orders for the given instrument.

getCash(*includeShort=True*)
Returns the available cash.

Parameters **includeShort** (*boolean.*) – Include cash from short positions.

getCommission()
Returns the strategy used to calculate order commissions.

Return type *Commission*.

getEquity()
Returns the portfolio value (cash + shares * price).

getFillStrategy()
Returns the *quantworks.broker.fillstrategy.FillStrategy* currently set.

getPositions()
Returns a dictionary that maps instruments to shares.

getShares(*instrument*)
Returns the number of shares for an instrument.

peekDateTime()
Return the datetime for the next event. This is needed to properly synchronize non-realtime subjects. Return None if this is a realtime subject.

setCommission(*commission*)
Sets the strategy to use to calculate order commissions.

Parameters **commission** (*Commission*.) – An object responsible for calculating order commissions.

setFillStrategy(*strategy*)
Sets the *quantworks.broker.fillstrategy.FillStrategy* to use.

setShares(*instrument*, *quantity*, *price*)
Set existing shares before the strategy starts executing.

Parameters

- **instrument** – Instrument identifier.
- **quantity** – The number of shares for the given instrument.
- **price** – The price for each share.

submitOrder(*order*)
Submits an order.

Parameters **order** (Order.) – The order to submit.

Note:

- After this call the order is in SUBMITTED state and an event is not triggered for this transition.
- Calling this twice on the same order will raise an exception.

---

class quantworks.broker.slippage.**SlippageModel**
Bases: object

Base class for slippage models.

Note: This is a base class and should not be used directly.

---

calculatePrice(*order*, *price*, *quantity*, *bar*, *volumeUsed*)
Returns the slipped price per share for an order.

Parameters

- **order** (*quantworks.broker.Order*.) – The order being filled.
- **price** (*float.*) – The price for each share before slippage.
- **quantity** (*float.*) – The amount of shares that will get filled at this time for this order.
- **bar** (*quantworks.bar.Bar*.) – The current bar.
- **volumeUsed** (*float.*) – The volume size that was taken so far from the current bar.

Return type float.

class quantworks.broker.slippage.**NoSlippage**
Bases: *quantworks.broker.slippage.SlippageModel*

A no slippage model.

calculatePrice(*order*, *price*, *quantity*, *bar*, *volumeUsed*)
Returns the slipped price per share for an order.

Parameters

- **order** (*quantworks.broker.Order*.) – The order being filled.
- **price** (*float.*) – The price for each share before slippage.
- **quantity** (*float.*) – The amount of shares that will get filled at this time for this order.
- **bar** (*quantworks.bar.Bar*.) – The current bar.
- **volumeUsed** (*float.*) – The volume size that was taken so far from the current bar.

Return type float.

class quantworks.broker.slippage.**VolumeShareSlippage**(*priceImpact=0.1*)

Bases: *quantworks.broker.slippage.SlippageModel*

A volume share slippage model as defined in Zipline's VolumeShareSlippage model. The slippage is calculated by multiplying the price impact constant by the square of the ratio of the order to the total volume.

Check https://www.quantopian.com/help#ide-slippage for more details.

Parameters **priceImpact** (*float.*) – Defines how large of an impact your order will have on the backtester's price calculation.

calculatePrice(*order*, *price*, *quantity*, *bar*, *volumeUsed*)

Returns the slipped price per share for an order.

Parameters

- **order** (*quantworks.broker.Order*.) – The order being filled.
- **price** (*float.*) – The price for each share before slippage.
- **quantity** (*float.*) – The amount of shares that will get filled at this time for this order.
- **bar** (*quantworks.bar.Bar*.) – The current bar.
- **volumeUsed** (*float.*) – The volume size that was taken so far from the current bar.

Return type float.

class quantworks.broker.fillstrategy.**FillStrategy**

Bases: object

Base class for order filling strategies for the backtester.

fillLimitOrder(*broker_*, *order*, *bar*)

Override to return the fill price and quantity for a limit order or None if the order can't be filled at the given time.

Parameters

- **broker** (Broker) – The broker.
- **order** (*quantworks.broker.LimitOrder*) – The order.
- **bar** (*quantworks.bar.Bar*) – The current bar.

Return type A FillInfo or None if the order should not be filled.

fillMarketOrder(*broker_*, *order*, *bar*)

Override to return the fill price and quantity for a market order or None if the order can't be filled at the given time.

Parameters

- **broker** (Broker) – The broker.

- **order** (*quantworks.broker.MarketOrder*) – The order.
- **bar** (*quantworks.bar.Bar*) – The current bar.

Return type A `FillInfo` or None if the order should not be filled.

fillStopLimitOrder(*broker_*, *order*, *bar*)
Override to return the fill price and quantity for a stop limit order or None if the order can't be filled at the given time.

Parameters

- **broker** (`Broker`) – The broker.
- **order** (*quantworks.broker.StopLimitOrder*) – The order.
- **bar** (*quantworks.bar.Bar*) – The current bar.

Return type A `FillInfo` or None if the order should not be filled.

fillStopOrder(*broker_*, *order*, *bar*)
Override to return the fill price and quantity for a stop order or None if the order can't be filled at the given time.

Parameters

- **broker** (`Broker`) – The broker.
- **order** (*quantworks.broker.StopOrder*) – The order.
- **bar** (*quantworks.bar.Bar*) – The current bar.

Return type A `FillInfo` or None if the order should not be filled.

onBars(*broker_*, *bars*)
Override (optional) to get notified when the broker is about to process new bars.

Parameters

- **broker** (`Broker`) – The broker.
- **bars** (*quantworks.bar.Bars*) – The current bars.

onOrderFilled(*broker_*, *order*)
Override (optional) to get notified when an order was filled, or partially filled.

Parameters

- **broker** (`Broker`) – The broker.
- **order** (*quantworks.broker.Order*) – The order filled.

class quantworks.broker.fillstrategy.**DefaultStrategy**(*volumeLimit=0.25*)
Bases: *quantworks.broker.fillstrategy.FillStrategy*

Default fill strategy.

Parameters **volumeLimit** (*float*) – The proportion of the volume that orders can take up in a bar. Must be > 0 and <= 1. If None, then volume limit is not checked.

This strategy works as follows:

- A *quantworks.broker.MarketOrder* is always filled using the open/close price.
- **A *quantworks.broker.LimitOrder* will be filled like this:**
  - If the limit price was penetrated with the open price, then the open price is used.
  - If the bar includes the limit price, then the limit price is used.

  - Note that when buying the price is penetrated if it gets <= the limit price, and when selling the price is penetrated if it gets >= the limit price

- **A *quantworks.broker.StopOrder* will be filled like this:**
  - If the stop price was penetrated with the open price, then the open price is used.
  - If the bar includes the stop price, then the stop price is used.
  - Note that when buying the price is penetrated if it gets >= the stop price, and when selling the price is penetrated if it gets <= the stop price
- **A *quantworks.broker.StopLimitOrder* will be filled like this:**
  - If the stop price was penetrated with the open price, or if the bar includes the stop price, then the limit order becomes active.
  - **If the limit order is active:**
    * If the limit order was activated in this same bar and the limit price is penetrated as well, then the best between the stop price and the limit fill price (as described earlier) is used.
    * If the limit order was activated at a previous bar then the limit fill price (as described earlier) is used.

Note:

- This is the default strategy used by the Broker.
- It uses *quantworks.broker.slippage.NoSlippage* slippage model by default.
- If volumeLimit is 0.25, and a certain bar's volume is 100, then no more than 25 shares can be used by all orders that get processed at that bar.
- If using trade bars, then all the volume from that bar can be used.

---

fillLimitOrder(*broker_*, *order*, *bar*)

Override to return the fill price and quantity for a limit order or None if the order can't be filled at the given time.

Parameters

- **broker** (Broker) – The broker.
- **order** (*quantworks.broker.LimitOrder*) – The order.
- **bar** (*quantworks.bar.Bar*) – The current bar.

Return type A FillInfo or None if the order should not be filled.

fillMarketOrder(*broker_*, *order*, *bar*)

Override to return the fill price and quantity for a market order or None if the order can't be filled at the given time.

Parameters

- **broker** (Broker) – The broker.
- **order** (*quantworks.broker.MarketOrder*) – The order.
- **bar** (*quantworks.bar.Bar*) – The current bar.

Return type A FillInfo or None if the order should not be filled.

fillStopLimitOrder(*broker_*, *order*, *bar*)
Override to return the fill price and quantity for a stop limit order or None if the order can't be filled at the given time.

Parameters

- **broker** (Broker) – The broker.
- **order** (*quantworks.broker.StopLimitOrder*) – The order.
- **bar** (*quantworks.bar.Bar*) – The current bar.

Return type A FillInfo or None if the order should not be filled.

fillStopOrder(*broker_*, *order*, *bar*)
Override to return the fill price and quantity for a stop order or None if the order can't be filled at the given time.

Parameters

- **broker** (Broker) – The broker.
- **order** (*quantworks.broker.StopOrder*) – The order.
- **bar** (*quantworks.bar.Bar*) – The current bar.

Return type A FillInfo or None if the order should not be filled.

onBars(*broker_*, *bars*)
Override (optional) to get notified when the broker is about to process new bars.

Parameters

- **broker** (Broker) – The broker.
- **bars** (*quantworks.bar.Bars*) – The current bars.

onOrderFilled(*broker_*, *order*)
Override (optional) to get notified when an order was filled, or partially filled.

Parameters

- **broker** (Broker) – The broker.
- **order** (*quantworks.broker.Order*) – The order filled.

setSlippageModel(*slippageModel*)
Set the slippage model to use.

Parameters **slippageModel** (*quantworks.broker.slippage.SlippageModel*) – The slippage model.

setVolumeLimit(*volumeLimit*)
Set the volume limit.

Parameters **volumeLimit** (*float*) – The proportion of the volume that orders can take up in a bar. Must be > 0 and <= 1. If None, then volume limit is not checked.

4.7 strategy – Basic strategy classes

Strategies are the classes that you define that implement the trading logic, when to buy, when to sell, etc.

Buying and selling can be done in two ways:

- Placing individual orders using any of the following methods:

- quantworks.strategy.BaseStrategy.marketOrder()
- quantworks.strategy.BaseStrategy.limitOrder()
- quantworks.strategy.BaseStrategy.stopOrder()
- quantworks.strategy.BaseStrategy.stopLimitOrder()
- Using a higher level interface that wrap a pair of entry/exit orders:
- quantworks.strategy.BaseStrategy.enterLong()
- quantworks.strategy.BaseStrategy.enterShort()
- quantworks.strategy.BaseStrategy.enterLongLimit()
- quantworks.strategy.BaseStrategy.enterShortLimit()

Positions are higher level abstractions for placing orders. They are escentially a pair of entry-exit orders and provide easier tracking for returns and PnL than using individual orders.

4.7.1 Strategy

4.7.2 Position

4.8 stratanalyzer – Strategy analyzers

Strategy analyzers provide an extensible way to attach different calculations to strategy executions.

class quantworks.stratanalyzer.**StrategyAnalyzer**
Bases: object

Base class for strategy analyzers.

Note: This is a base class and should not be used directly.

4.8.1 Returns

4.8.2 Sharpe Ratio

4.8.3 DrawDown

class quantworks.stratanalyzer.drawdown.**DrawDown**
Bases: *quantworks.stratanalyzer.StrategyAnalyzer*

A *quantworks.stratanalyzer.StrategyAnalyzer* that calculates max. drawdown and longest drawdown duration for the portfolio.

getLongestDrawDownDuration()
Returns the duration of the longest drawdown.

Return type datetime.timedelta.

Note: Note that this is the duration of the longest drawdown, not necessarily the deepest one.

getMaxDrawDown()
Returns the max. (deepest) drawdown.

4.8.4 Trades

4.8.5 Example

Save this code as sma_crossover.py:

```
from quantworks import strategy
from quantworks.technical import ma
from quantworks.technical import cross


class SMACrossOver(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument, smaPeriod):
        super(SMACrossOver, self).__init__(feed)
        self.__instrument = instrument
        self.__position = None
        # We'll use adjusted close values instead of regular close values.
        self.setUseAdjustedValues(True)
        self.__prices = feed[instrument].getPriceDataSeries()
        self.__sma = ma.SMA(self.__prices, smaPeriod)

    def getSMA(self):
        return self.__sma

    def onEnterCanceled(self, position):
        self.__position = None

    def onExitOk(self, position):
        self.__position = None

    def onExitCanceled(self, position):
        # If the exit was canceled, re-submit it.
        self.__position.exitMarket()

    def onBars(self, bars):
        # If a position was not opened, check if we should enter a long position.
        if self.__position is None:
            if cross.cross_above(self.__prices, self.__sma) > 0:
                shares = int(self.getBroker().getCash() * 0.9 / bars[self.__
↪instrument].getPrice())
                # Enter a buy market order. The order is good till canceled.
                self.__position = self.enterLong(self.__instrument, shares, True)
        # Check if we have to exit the position.
        elif not self.__position.exitActive() and cross.cross_below(self.__prices,␣
↪self.__sma) > 0:
            self.__position.exitMarket()
```

and save this code in a different file:

```
from __future__ import print_function

from quantworks.barfeed import yahoofeed
from quantworks.stratanalyzer import returns
```

(continues on next page)

(continued from previous page)

```
from quantworks.stratanalyzer import sharpe
from quantworks.stratanalyzer import drawdown
from quantworks.stratanalyzer import trades

from . import sma_crossover

# Load the bars. This file was manually downloaded from Yahoo Finance.
feed = yahoofeed.Feed()
feed.addBarsFromCSV("orcl", "orcl-2000-yahoofinance.csv")

# Evaluate the strategy with the feed's bars.
myStrategy = sma_crossover.SMACrossOver(feed, "orcl", 20)

# Attach different analyzers to a strategy before executing it.
retAnalyzer = returns.Returns()
myStrategy.attachAnalyzer(retAnalyzer)
sharpeRatioAnalyzer = sharpe.SharpeRatio()
myStrategy.attachAnalyzer(sharpeRatioAnalyzer)
drawDownAnalyzer = drawdown.DrawDown()
myStrategy.attachAnalyzer(drawDownAnalyzer)
tradesAnalyzer = trades.Trades()
myStrategy.attachAnalyzer(tradesAnalyzer)

# Run the strategy.
myStrategy.run()

print("Final portfolio value: $%.2f" % myStrategy.getResult())
print("Cumulative returns: %.2f %%" % (retAnalyzer.getCumulativeReturns()[-1] * 100))
print("Sharpe ratio: %.2f" % (sharpeRatioAnalyzer.getSharpeRatio(0.05)))
print("Max. drawdown: %.2f %%" % (drawDownAnalyzer.getMaxDrawDown() * 100))
print("Longest drawdown duration: %s" % (drawDownAnalyzer.
↪getLongestDrawDownDuration()))

print("")
print("Total trades: %d" % (tradesAnalyzer.getCount()))
if tradesAnalyzer.getCount() > 0:
    profits = tradesAnalyzer.getAll()
    print("Avg. profit: $%2.f" % (profits.mean()))
    print("Profits std. dev.: $%2.f" % (profits.std()))
    print("Max. profit: $%2.f" % (profits.max()))
    print("Min. profit: $%2.f" % (profits.min()))
    returns = tradesAnalyzer.getAllReturns()
    print("Avg. return: %2.f %%" % (returns.mean() * 100))
    print("Returns std. dev.: %2.f %%" % (returns.std() * 100))
    print("Max. return: %2.f %%" % (returns.max() * 100))
    print("Min. return: %2.f %%" % (returns.min() * 100))

print("")
print("Profitable trades: %d" % (tradesAnalyzer.getProfitableCount()))
if tradesAnalyzer.getProfitableCount() > 0:
    profits = tradesAnalyzer.getProfits()
    print("Avg. profit: $%2.f" % (profits.mean()))
    print("Profits std. dev.: $%2.f" % (profits.std()))
    print("Max. profit: $%2.f" % (profits.max()))
    print("Min. profit: $%2.f" % (profits.min()))
    returns = tradesAnalyzer.getPositiveReturns()
    print("Avg. return: %2.f %%" % (returns.mean() * 100))
```

(continues on next page)

(continued from previous page)

```
    print("Returns std. dev.: %2.f %%" % (returns.std() * 100))
    print("Max. return: %2.f %%" % (returns.max() * 100))
    print("Min. return: %2.f %%" % (returns.min() * 100))

print("")
print("Unprofitable trades: %d" % (tradesAnalyzer.getUnprofitableCount()))
if tradesAnalyzer.getUnprofitableCount() > 0:
    losses = tradesAnalyzer.getLosses()
    print("Avg. loss: $%2.f" % (losses.mean()))
    print("Losses std. dev.: $%2.f" % (losses.std()))
    print("Max. loss: $%2.f" % (losses.min()))
    print("Min. loss: $%2.f" % (losses.max()))
    returns = tradesAnalyzer.getNegativeReturns()
    print("Avg. return: %2.f %%" % (returns.mean() * 100))
    print("Returns std. dev.: %2.f %%" % (returns.std() * 100))
    print("Max. return: %2.f %%" % (returns.max() * 100))
    print("Min. return: %2.f %%" % (returns.min() * 100))
```

The output should look like this:

```
Final portfolio value: $1295462.60
Cumulative returns: 29.55 %
Sharpe ratio: 0.70
Max. drawdown: 24.58 %
Longest drawdown duration: 277 days, 0:00:00

Total trades: 13
Avg. profit: $14391
Profits std. dev.: $127520
Max. profit: $420782
Min. profit: $-89317
Avg. return:  2 %
Returns std. dev.: 13 %
Max. return: 46 %
Min. return: -7 %

Profitable trades: 3
Avg. profit: $196972
Profits std. dev.: $158985
Max. profit: $420782
Min. profit: $66466
Avg. return: 21 %
Returns std. dev.: 18 %
Max. return: 46 %
Min. return:  6 %

Unprofitable trades: 10
Avg. loss: $-40383
Losses std. dev.: $23579
Max. loss: $-89317
Min. loss: $-4702
Avg. return: -3 %
Returns std. dev.:  2 %
Max. return: -0 %
Min. return: -7 %
```

4.9 plotter – Strategy plotter

4.10 optimizer – Parallel optimizers

class quantworks.optimizer.server.**Results**(*parameters*, *result*)
Bases: object

The results of the strategy executions.

getParameters()
Returns a sequence of parameter values.

getResult()
Returns the result for a given set of parameters.

quantworks.optimizer.server.**serve**(*barFeed*, *strategyParameters*, *address*, *port*, *batchSize=200*)
Executes a server that will provide bars and strategy parameters for workers to use.

Parameters

- **barFeed** (quantworks.barfeed.BarFeed.) – The bar feed that each worker will use to backtest the strategy.
- **strategyParameters** – The set of parameters to use for backtesting. An iterable object where **each element is a tuple that holds parameter values**.
- **address** (*string.*) – The address to listen for incoming worker connections.
- **port** (*int.*) – The port to listen for incoming worker connections.
- **batchSize** (*int.*) – The number of strategy executions that are delivered to each worker.

Return type A *Results* instance with the best results found or None if no results were obtained.

Note:

- The server component will split strategy executions in chunks which are distributed among the different workers. You can optionally set the chunk size by passing in **batchSize** to the constructor of **quantworks.optimizer.xmlrpcserver.Server**.
- The quantworks.strategy.BaseStrategy.getResult() method is used to select the best strategy execution. You can override that method to rank executions using a different criteria.

4.11 marketsession – Market sessions

class quantworks.marketsession.**MarketSession**
Bases: object

Base class for market sessions.

Note: This is a base class and should not be used directly.

---

classmethod getTimezone()
Returns the pytz timezone for the market session.

class quantworks.marketsession.**NASDAQ**
Bases: *quantworks.marketsession.MarketSession*

NASDAQ market session.

class quantworks.marketsession.**NYSE**
Bases: *quantworks.marketsession.MarketSession*

New York Stock Exchange market session.

class quantworks.marketsession.**USEquities**
Bases: *quantworks.marketsession.MarketSession*

US Equities market session.

class quantworks.marketsession.**MERVAL**
Bases: *quantworks.marketsession.MarketSession*

Buenos Aires (Argentina) market session.

class quantworks.marketsession.**BOVESPA**
Bases: *quantworks.marketsession.MarketSession*

BOVESPA (Brazil) market session.

class quantworks.marketsession.**FTSE**
Bases: *quantworks.marketsession.MarketSession*

London Stock Exchange market session.

class quantworks.marketsession.**TSE**
Bases: *quantworks.marketsession.MarketSession*

Tokyo Stock Exchange market session.

CHAPTER 5

Tools

5.1 Quandl

5.2 BarFeed resampling

CHAPTER 6

Event profiler

Inspired in QSTK, the **eventprofiler** module is a tool to analyze, statistically, how events affect future equity prices. The event profiler scans over historical data for a specified event and then calculates the impact of that event on the equity prices in the past and the future over a certain lookback period.

The goal of this tool is to help you quickly validate an idea, before moving forward with the backtesting process.

6.1 Example

The following example is inspired on the 'Buy-on-Gap Model' from Ernie Chan's book: 'Algorithmic Trading: Winning Strategies and Their Rationale':

- The idea is to select a stock near the market open whose returns from their previous day's lows to today's open are lower that one standard deviation. The standard deviation is computed using the daily close-to-close returns of the last 90 days. These are the stocks that "gapped down".
- This is narrowed down by requiring the open price to be higher than the 20-day moving average of the closing price.

```
from __future__ import print_function

from quantworks import eventprofiler
from quantworks.technical import stats
from quantworks.technical import roc
from quantworks.technical import ma
from quantworks.tools import quandl

# Event inspired on an example from Ernie Chan's book:
# 'Algorithmic Trading: Winning Strategies and Their Rationale'


class BuyOnGap(eventprofiler.Predicate):
    def __init__(self, feed):
        super(BuyOnGap, self).__init__()
```

(continues on next page)

(continued from previous page)

```
        stdDevPeriod = 90
        smaPeriod = 20
        self.__returns = {}
        self.__stdDev = {}
        self.__ma = {}
        for instrument in feed.getRegisteredInstruments():
            priceDS = feed[instrument].getAdjCloseDataSeries()
            # Returns over the adjusted close values.
            self.__returns[instrument] = roc.RateOfChange(priceDS, 1)
            # StdDev over those returns.
            self.__stdDev[instrument] = stats.StdDev(self.__returns[instrument],␣
↪stdDevPeriod)
            # MA over the adjusted close values.
            self.__ma[instrument] = ma.SMA(priceDS, smaPeriod)

    def __gappedDown(self, instrument, bards):
        ret = False
        if self.__stdDev[instrument][-1] is not None:
            prevBar = bards[-2]
            currBar = bards[-1]
            low2OpenRet = (currBar.getOpen(True) - prevBar.getLow(True)) /␣
↪float(prevBar.getLow(True))
            if low2OpenRet < (self.__returns[instrument][-1] - self.__
↪stdDev[instrument][-1]):
                ret = True
        return ret

    def __aboveSMA(self, instrument, bards):
        ret = False
        if self.__ma[instrument][-1] is not None and bards[-1].getOpen(True) > self.__
↪ma[instrument][-1]:
            ret = True
        return ret

    def eventOccurred(self, instrument, bards):
        ret = False
        if self.__gappedDown(instrument, bards) and self.__aboveSMA(instrument,␣
↪bards):
            ret = True
        return ret


def main(plot):
    instruments = ["IBM", "AES", "AIG"]
    feed = quandl.build_feed("WIKI", instruments, 2008, 2009, ".")

    predicate = BuyOnGap(feed)
    eventProfiler = eventprofiler.Profiler(predicate, 5, 5)
    eventProfiler.run(feed, True)

    results = eventProfiler.getResults()
    print("%d events found" % (results.getEventCount()))
    if plot:
        eventprofiler.plot(results)

if __name__ == "__main__":
```

(continues on next page)

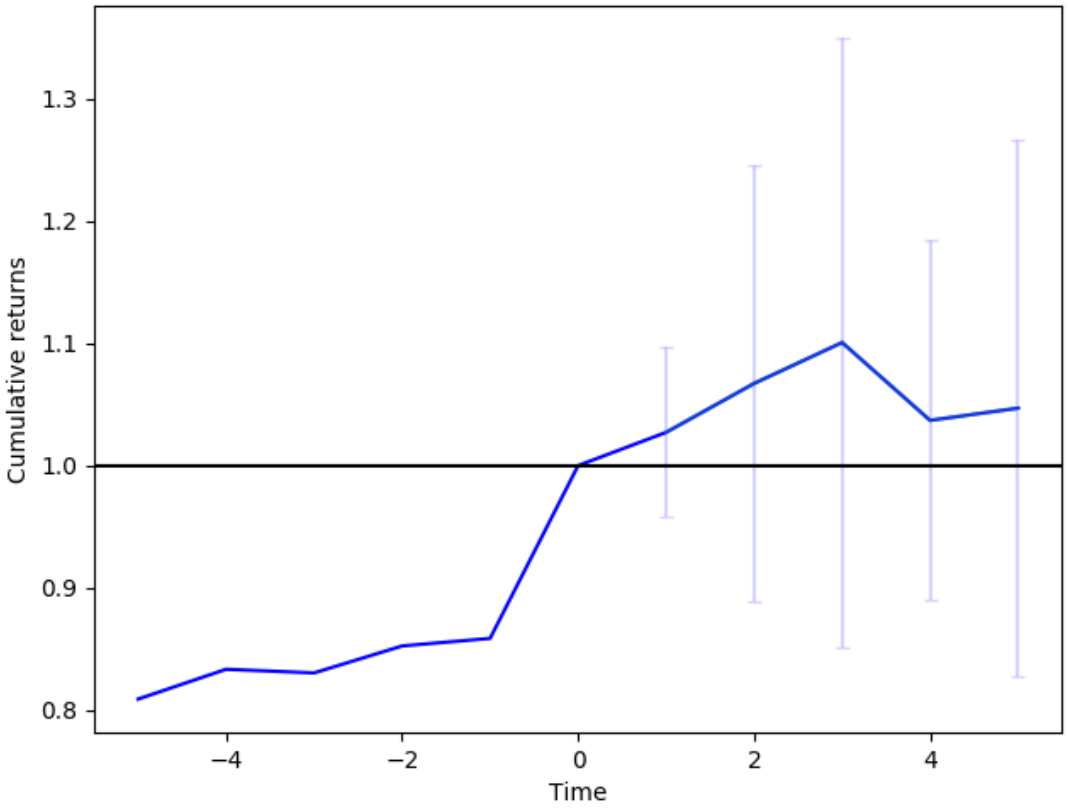
(continued from previous page)

```
    main(True)
```

The code is doing 4 things:

1. Declaring a `Predicate` that implements the 'Buy-on-Gap Model' event identification.
2. Loading bars for some stocks.
3. Running the analysis.
4. Plotting the results.

This is what the output should look like:




```
2017-07-22 00:26:06,574 quandl [INFO] Downloading IBM 2008 to ./WIKI-IBM-2008-quandl.
↪csv
2017-07-22 00:26:08,299 quandl [INFO] Downloading AES 2008 to ./WIKI-AES-2008-quandl.
↪csv
2017-07-22 00:26:09,849 quandl [INFO] Downloading AIG 2008 to ./WIKI-AIG-2008-quandl.
↪csv
2017-07-22 00:26:11,513 quandl [INFO] Downloading IBM 2009 to ./WIKI-IBM-2009-quandl.
↪csv
2017-07-22 00:26:13,128 quandl [INFO] Downloading AES 2009 to ./WIKI-AES-2009-quandl.
↪csv
2017-07-22 00:26:14,626 quandl [INFO] Downloading AIG 2009 to ./WIKI-AIG-2009-quandl.
↪csv
```

(continues on next page)

(continued from previous page)

```
15 events found
```

Note that **Cummulative returns are normalized to the time of the event**.

CHAPTER 7

TA-Lib integration

The **quantworks.talibext.indicator** module provides integration with Python wrapper for TA-Lib (http://mrjbq7.github.com/ta-lib/) to enable calling TA-Lib functions directly with `quantworks.dataseries.DataSeries` or `quantworks.dataseries.bards.BarDataSeries` instances instead of numpy arrays.

If you're familiar with the **talib** module, then using the **quantworks.talibext.indicator** module should be straightforward. When using **talib** standalone you do something like this:

```
import numpy
import talib

data = numpy.random.random(100)
upper, middle, lower = talib.BBANDS(data, matype=talib.MA_T3)
```

To use the **quantworks.talibext.indicator** module in your strategies you should do something like this:

```
def onBars(self, bars):
    closeDs = self.getFeed().getDataSeries("orcl").getCloseDataSeries()
    upper, middle, lower = quantworks.talibext.indicator.BBANDS(closeDs, 100,␣
↪matype=talib.MA_T3)
    if upper != None:
        print "%s" % upper[-1]
```

Every function in the **quantworks.talibext.indicator** module receives one or more dataseries (most receive just one) and the number of values to use from the dataseries. In the example above, we're calculating Bollinger Bands over the last 100 closing prices.

If the parameter name is **ds**, then you should pass a regular `quantworks.dataseries.DataSeries` instance, like the one shown in the example above.

If the parameter name is **barDs**, then you should pass a `quantworks.dataseries.bards.BarDataSeries` instance, like in the next example:

```
def onBars(self, bars):
    barDs = self.getFeed().getDataSeries("orcl")
    sar = indicator.SAR(barDs, 100)
```

(continues on next page)

(continued from previous page)

```
    if sar != None:
        print "%s" % sar[-1]
```

The following TA-Lib functions are available through the **quantworks.talibext.indicator** module:

CHAPTER 8

Computational Investing Part I

As I was taking the Computational Investing Part I course in 2012 I had to work on a set of assignments and for one of them I used QuantWorks.

8.1 Homework 1

For this assignment I had to pick 4 stocks, invest a total of $1000000 during 2011, and calculate:

- Final portfolio value
- Anual return
- Average daily return
- Std. dev. of daily returns
- Sharpe ratio

Download the data with the following commands:

```
python -m "quantworks.tools.quandl" --source-code="WIKI" --table-code="IBM" --from-
↪year=2011 --to-year=2011 --storage=. --force-download --frequency=daily
python -m "quantworks.tools.quandl" --source-code="WIKI" --table-code="AES" --from-
↪year=2011 --to-year=2011 --storage=. --force-download --frequency=daily
python -m "quantworks.tools.quandl" --source-code="WIKI" --table-code="AIG" --from-
↪year=2011 --to-year=2011 --storage=. --force-download --frequency=daily
python -m "quantworks.tools.quandl" --source-code="WIKI" --table-code="ORCL" --from-
↪year=2011 --to-year=2011 --storage=. --force-download --frequency=daily
```

Although the deliverable was an Excel spreadsheet, I validated the results using this piece of code:

```
from __future__ import print_function

from quantworks import strategy
from quantworks.barfeed import quandlfeed
```

(continues on next page)

(continued from previous page)

```
from quantworks.stratanalyzer import returns
from quantworks.stratanalyzer import sharpe
from quantworks.utils import stats


class MyStrategy(strategy.BacktestingStrategy):
    def __init__(self, feed):
        super(MyStrategy, self).__init__(feed, 1000000)

        # We wan't to use adjusted close prices instead of close.
        self.setUseAdjustedValues(True)

        # Place the orders to get them processed on the first bar.
        orders = {
            "ibm": 1996,
            "aes": 22565,
            "aig": 5445,
            "orcl": 8582,
        }
        for instrument, quantity in orders.items():
            self.marketOrder(instrument, quantity, onClose=True, allOrNone=True)

    def onBars(self, bars):
        pass

# Load the bar feed from the CSV file
feed = quandlfeed.Feed()
feed.addBarsFromCSV("ibm", "WIKI-IBM-2011-quandl.csv")
feed.addBarsFromCSV("aes", "WIKI-AES-2011-quandl.csv")
feed.addBarsFromCSV("aig", "WIKI-AIG-2011-quandl.csv")
feed.addBarsFromCSV("orcl", "WIKI-ORCL-2011-quandl.csv")

# Evaluate the strategy with the feed's bars.
myStrategy = MyStrategy(feed)

# Attach returns and sharpe ratio analyzers.
retAnalyzer = returns.Returns()
myStrategy.attachAnalyzer(retAnalyzer)
sharpeRatioAnalyzer = sharpe.SharpeRatio()
myStrategy.attachAnalyzer(sharpeRatioAnalyzer)

# Run the strategy
myStrategy.run()

# Print the results.
print("Final portfolio value: $%.2f" % myStrategy.getResult())
print("Anual return: %.2f %%" % (retAnalyzer.getCumulativeReturns()[-1] * 100))
print("Average daily return: %.2f %%" % (stats.mean(retAnalyzer.getReturns()) * 100))
print("Std. dev. daily return: %.4f" % (stats.stddev(retAnalyzer.getReturns())))
print("Sharpe ratio: %.2f" % (sharpeRatioAnalyzer.getSharpeRatio(0)))
```

The results were:

```
Final portfolio value: $876350.83
Anual return: -12.36 %
Average daily return: -0.04 %
Std. dev. daily return: 0.0176
```

(continues on next page)

(continued from previous page)

```
Sharpe ratio: -0.33
```

CHAPTER 9

Sample strategies

9.1 Momentum

9.1.1 VWAP Momentum Trade

This example is based on Momentum Trade

```
from __future__ import print_function

from quantworks import strategy
from quantworks import plotter
from quantworks.tools import quandl
from quantworks.technical import vwap
from quantworks.stratanalyzer import sharpe


class VWAPMomentum(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument, vwapWindowSize, threshold):
        super(VWAPMomentum, self).__init__(feed)
        self.__instrument = instrument
        self.__vwap = vwap.VWAP(feed[instrument], vwapWindowSize)
        self.__threshold = threshold

    def getVWAP(self):
        return self.__vwap

    def onBars(self, bars):
        vwap = self.__vwap[-1]
        if vwap is None:
            return

        shares = self.getBroker().getShares(self.__instrument)
        price = bars[self.__instrument].getClose()
        notional = shares * price
```

(continues on next page)

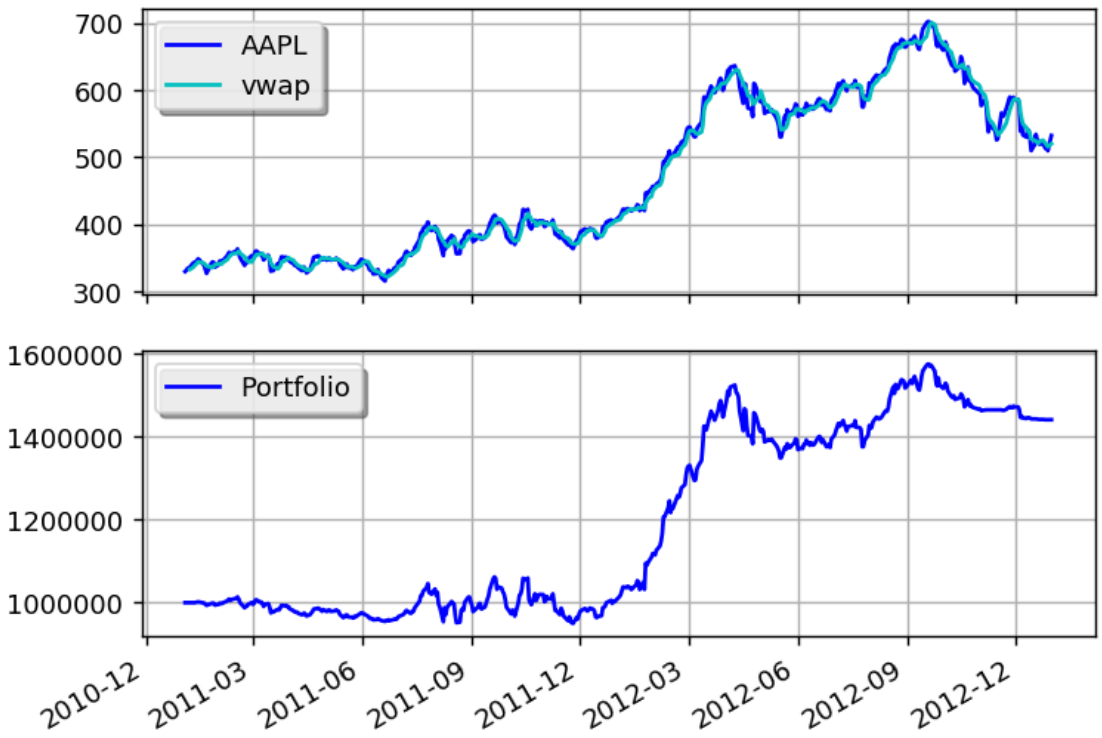
(continued from previous page)

```
        if price > vwap * (1 + self.__threshold) and notional < 1000000:
            self.marketOrder(self.__instrument, 100)
        elif price < vwap * (1 - self.__threshold) and notional > 0:
            self.marketOrder(self.__instrument, -100)


def main(plot):
    instrument = "AAPL"
    vwapWindowSize = 5
    threshold = 0.01

    # Download the bars.
    feed = quandl.build_feed("WIKI", [instrument], 2011, 2012, ".")

    strat = VWAPMomentum(feed, instrument, vwapWindowSize, threshold)
    sharpeRatioAnalyzer = sharpe.SharpeRatio()
    strat.attachAnalyzer(sharpeRatioAnalyzer)

    if plot:
        plt = plotter.StrategyPlotter(strat, True, False, True)
        plt.getInstrumentSubplot(instrument).addDataSeries("vwap", strat.getVWAP())

    strat.run()
    print("Sharpe ratio: %.2f" % sharpeRatioAnalyzer.getSharpeRatio(0.05))

    if plot:
        plt.plot()


if __name__ == "__main__":
    main(True)
```

this is what the output should look like:

```
2017-07-24 22:43:34,182 quandl [INFO] Downloading AAPL 2011 to ./WIKI-AAPL-2011-
↪quandl.csv
2017-07-24 22:43:36,415 quandl [INFO] Downloading AAPL 2012 to ./WIKI-AAPL-2012-
↪quandl.csv
Sharpe ratio: 0.89
```

and this is what the plot should look like:



You can get better returns by tunning the VWAP and threshold parameters.

9.1.2 SMA Crossover

Save this code as sma_crossover.py:

```
from quantworks import strategy
from quantworks.technical import ma
from quantworks.technical import cross


class SMACrossOver(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument, smaPeriod):
        super(SMACrossOver, self).__init__(feed)
        self.__instrument = instrument
        self.__position = None
        # We'll use adjusted close values instead of regular close values.
        self.setUseAdjustedValues(True)
        self.__prices = feed[instrument].getPriceDataSeries()
        self.__sma = ma.SMA(self.__prices, smaPeriod)

    def getSMA(self):
        return self.__sma

    def onEnterCanceled(self, position):
        self.__position = None

    def onExitOk(self, position):
        self.__position = None
```

(continues on next page)

(continued from previous page)

```
    def onExitCanceled(self, position):
        # If the exit was canceled, re-submit it.
        self.__position.exitMarket()

    def onBars(self, bars):
        # If a position was not opened, check if we should enter a long position.
        if self.__position is None:
            if cross.cross_above(self.__prices, self.__sma) > 0:
                shares = int(self.getBroker().getCash() * 0.9 / bars[self.__
↪instrument].getPrice())
                # Enter a buy market order. The order is good till canceled.
                self.__position = self.enterLong(self.__instrument, shares, True)
        # Check if we have to exit the position.
        elif not self.__position.exitActive() and cross.cross_below(self.__prices,␣
↪self.__sma) > 0:
            self.__position.exitMarket()
```

and use the following code to execute the strategy:

```
from __future__ import print_function

import sma_crossover
from quantworks import plotter
from quantworks.tools import quandl
from quantworks.stratanalyzer import sharpe


def main(plot):
    instrument = "AAPL"
    smaPeriod = 163

    # Download the bars.
    feed = quandl.build_feed("WIKI", [instrument], 2011, 2012, ".")

    strat = sma_crossover.SMACrossOver(feed, instrument, smaPeriod)
    sharpeRatioAnalyzer = sharpe.SharpeRatio()
    strat.attachAnalyzer(sharpeRatioAnalyzer)

    if plot:
        plt = plotter.StrategyPlotter(strat, True, False, True)
        plt.getInstrumentSubplot(instrument).addDataSeries("sma", strat.getSMA())

    strat.run()
    print("Sharpe ratio: %.2f" % sharpeRatioAnalyzer.getSharpeRatio(0.05))

    if plot:
        plt.plot()


if __name__ == "__main__":
    main(True)
```

This is what the output should look like:

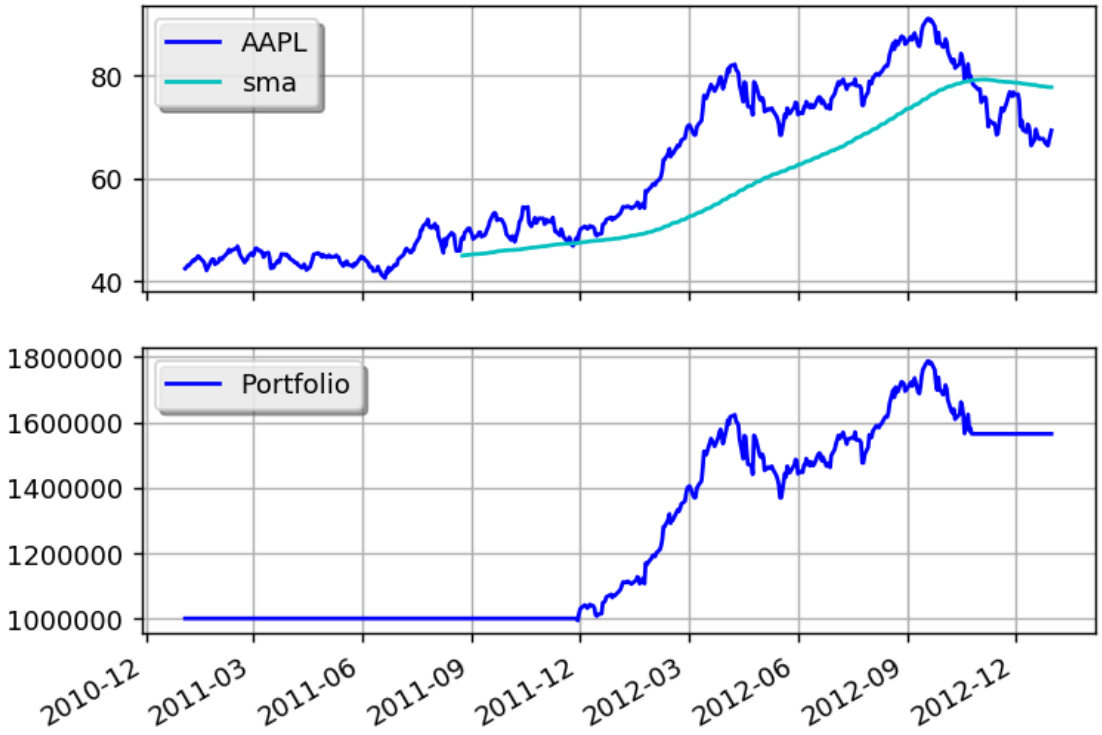
```
2017-07-24 22:56:58,112 quandl [INFO] Downloading AAPL 2011 to ./WIKI-AAPL-2011-
↪quandl.csv
2017-07-24 22:57:02,364 quandl [INFO] Downloading AAPL 2012 to ./WIKI-AAPL-2012-
↪quandl.csv
```

(continues on next page)

(continued from previous page)

```
Sharpe ratio: 1.12
```

and this is what the plot should look like:



You can get better returns by tunning the sma period.

9.1.3 Market Timing Using Moving-Average Crossovers

This example is inspired on the Market Timing / GTAA model described in:

- http://mebfaber.com/timing-model/
- http://papers.ssrn.com/sol3/papers.cfm?abstract_id=962461

The stragegy supports analyzing more than one instrument per asset class, and selects the one that has highest returns in the last month.

```
from __future__ import print_function

from quantworks import strategy
from quantworks import plotter
from quantworks.barfeed import yahoofeed
from quantworks.technical import ma
from quantworks.technical import cumret
from quantworks.stratanalyzer import sharpe
from quantworks.stratanalyzer import returns


class MarketTiming(strategy.BacktestingStrategy):
    def __init__(self, feed, instrumentsByClass, initialCash):
```

(continues on next page)

(continued from previous page)

```
        super(MarketTiming, self).__init__(feed, initialCash)
        self.setUseAdjustedValues(True)
        self.__instrumentsByClass = instrumentsByClass
        self.__rebalanceMonth = None
        self.__sharesToBuy = {}
        # Initialize indicators for each instrument.
        self.__sma = {}
        for assetClass in instrumentsByClass:
            for instrument in instrumentsByClass[assetClass]:
                priceDS = feed[instrument].getPriceDataSeries()
                self.__sma[instrument] = ma.SMA(priceDS, 200)

    def _shouldRebalance(self, dateTime):
        return dateTime.month != self.__rebalanceMonth

    def _getRank(self, instrument):
        # If the price is below the SMA, then this instrument doesn't rank at
        # all.
        smas = self.__sma[instrument]
        price = self.getLastPrice(instrument)
        if len(smas) == 0 or smas[-1] is None or price < smas[-1]:
            return None

        # Rank based on 20 day returns.
        ret = None
        lookBack = 20
        priceDS = self.getFeed()[instrument].getPriceDataSeries()
        if len(priceDS) >= lookBack and smas[-1] is not None and smas[-1*lookBack] is␣
↪not None:
            ret = (priceDS[-1] - priceDS[-1*lookBack]) / float(priceDS[-1*lookBack])
        return ret

    def _getTopByClass(self, assetClass):
        # Find the instrument with the highest rank.
        ret = None
        highestRank = None
        for instrument in self.__instrumentsByClass[assetClass]:
            rank = self._getRank(instrument)
            if rank is not None and (highestRank is None or rank > highestRank):
                highestRank = rank
                ret = instrument
        return ret

    def _getTop(self):
        ret = {}
        for assetClass in self.__instrumentsByClass:
            ret[assetClass] = self._getTopByClass(assetClass)
        return ret

    def _placePendingOrders(self):
        # Use less chash just in case price changes too much.
        remainingCash = round(self.getBroker().getCash() * 0.9, 2)

        for instrument in self.__sharesToBuy:
            orderSize = self.__sharesToBuy[instrument]
            if orderSize > 0:
                # Adjust the order size based on available cash.
```

(continues on next page)

(continued from previous page)

```
                lastPrice = self.getLastPrice(instrument)
                cost = orderSize * lastPrice
                while cost > remainingCash and orderSize > 0:
                    orderSize -= 1
                    cost = orderSize * lastPrice
                if orderSize > 0:
                    remainingCash -= cost
                    assert(remainingCash >= 0)

            if orderSize != 0:
                self.info("Placing market order for %d %s shares" % (orderSize,␣
↪instrument))
                self.marketOrder(instrument, orderSize, goodTillCanceled=True)
                self.__sharesToBuy[instrument] -= orderSize

    def _logPosSize(self):
        totalEquity = self.getBroker().getEquity()
        positions = self.getBroker().getPositions()
        for instrument in self.getBroker().getPositions():
            posSize = positions[instrument] * self.getLastPrice(instrument) /␣
↪totalEquity * 100
            self.info("%s - %0.2f %%" % (instrument, posSize))

    def _rebalance(self):
        self.info("Rebalancing")

        # Cancel all active/pending orders.
        for order in self.getBroker().getActiveOrders():
            self.getBroker().cancelOrder(order)

        cashPerAssetClass = round(self.getBroker().getEquity() / float(len(self.__
↪instrumentsByClass)), 2)
        self.__sharesToBuy = {}

        # Calculate which positions should be open during the next period.
        topByClass = self._getTop()
        for assetClass in topByClass:
            instrument = topByClass[assetClass]
            self.info("Best for class %s: %s" % (assetClass, instrument))
            if instrument is not None:
                lastPrice = self.getLastPrice(instrument)
                cashForInstrument = round(cashPerAssetClass - self.getBroker().
↪getShares(instrument) * lastPrice, 2)
                # This may yield a negative value and we have to reduce this
                # position.
                self.__sharesToBuy[instrument] = int(cashForInstrument / lastPrice)

        # Calculate which positions should be closed.
        for instrument in self.getBroker().getPositions():
            if instrument not in topByClass.values():
                currentShares = self.getBroker().getShares(instrument)
                assert(instrument not in self.__sharesToBuy)
                self.__sharesToBuy[instrument] = currentShares * -1

    def getSMA(self, instrument):
        return self.__sma[instrument]
```

(continues on next page)

(continued from previous page)

```
    def onBars(self, bars):
        currentDateTime = bars.getDateTime()

        if self._shouldRebalance(currentDateTime):
            self.__rebalanceMonth = currentDateTime.month
            self._rebalance()

        self._placePendingOrders()


def main(plot):
    initialCash = 10000
    instrumentsByClass = {
        "US Stocks": ["VTI"],
        "Foreign Stocks": ["VEU"],
        "US 10 Year Government Bonds": ["IEF"],
        "Real Estate": ["VNQ"],
        "Commodities": ["DBC"],
    }

    # Load the bars. These files were manually downloaded from Yahoo Finance.
    feed = yahoofeed.Feed()
    instruments = ["SPY"]
    for assetClass in instrumentsByClass:
        instruments.extend(instrumentsByClass[assetClass])

    for year in range(2007, 2013+1):
        for instrument in instruments:
            fileName = "%s-%d-yahoofinance.csv" % (instrument, year)
            print("Loading bars from %s" % fileName)
            feed.addBarsFromCSV(instrument, fileName)

    # Build the strategy and attach some metrics.
    strat = MarketTiming(feed, instrumentsByClass, initialCash)
    sharpeRatioAnalyzer = sharpe.SharpeRatio()
    strat.attachAnalyzer(sharpeRatioAnalyzer)
    returnsAnalyzer = returns.Returns()
    strat.attachAnalyzer(returnsAnalyzer)

    if plot:
        plt = plotter.StrategyPlotter(strat, False, False, True)
        plt.getOrCreateSubplot("cash").addCallback("Cash", lambda x: strat.
↪getBroker().getCash())
        # Plot strategy vs. SPY cumulative returns.
        plt.getOrCreateSubplot("returns").addDataSeries("SPY", cumret.
↪CumulativeReturn(feed["SPY"].getPriceDataSeries()))
        plt.getOrCreateSubplot("returns").addDataSeries("Strategy", returnsAnalyzer.
↪getCumulativeReturns())

    strat.run()
    print("Sharpe ratio: %.2f" % sharpeRatioAnalyzer.getSharpeRatio(0.05))
    print("Returns: %.2f %%" % (returnsAnalyzer.getCumulativeReturns()[-1] * 100))

    if plot:
        plt.plot()
```

(continues on next page)

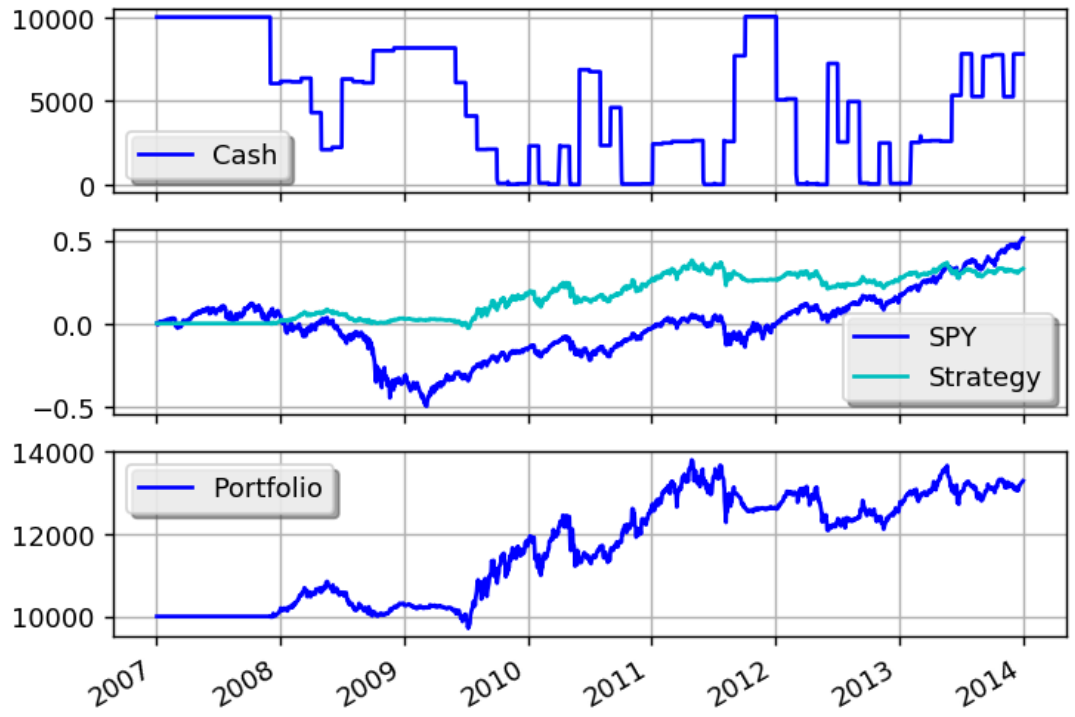
(continued from previous page)

```
if __name__ == "__main__":
    main(True)
```

This is what the output should look like:

```
Loading bars from SPY-2007-yahoofinance.csv
Loading bars from VTI-2007-yahoofinance.csv
Loading bars from DBC-2007-yahoofinance.csv
Loading bars from IEF-2007-yahoofinance.csv
Loading bars from VEU-2007-yahoofinance.csv
Loading bars from VNQ-2007-yahoofinance.csv
Loading bars from SPY-2008-yahoofinance.csv
Loading bars from VTI-2008-yahoofinance.csv
Loading bars from DBC-2008-yahoofinance.csv
Loading bars from IEF-2008-yahoofinance.csv
.
.
.
2013-10-01 00:00:00 strategy [INFO] Best for class US Stocks: VTI
2013-10-01 00:00:00 strategy [INFO] Best for class Commodities: None
2013-10-01 00:00:00 strategy [INFO] Best for class US 10 Year Government Bonds: None
2013-10-01 00:00:00 strategy [INFO] Best for class Foreign Stocks: VEU
2013-10-01 00:00:00 strategy [INFO] Best for class Real Estate: None
2013-10-01 00:00:00 strategy [INFO] Placing market order for -2 VEU shares
2013-11-01 00:00:00 strategy [INFO] Rebalancing
2013-11-01 00:00:00 strategy [INFO] Best for class US Stocks: VTI
2013-11-01 00:00:00 strategy [INFO] Best for class Commodities: None
2013-11-01 00:00:00 strategy [INFO] Best for class US 10 Year Government Bonds: None
2013-11-01 00:00:00 strategy [INFO] Best for class Foreign Stocks: VEU
2013-11-01 00:00:00 strategy [INFO] Best for class Real Estate: VNQ
2013-11-01 00:00:00 strategy [INFO] Placing market order for -1 VTI shares
2013-11-01 00:00:00 strategy [INFO] Placing market order for -1 VEU shares
2013-11-01 00:00:00 strategy [INFO] Placing market order for 39 VNQ shares
2013-12-02 00:00:00 strategy [INFO] Rebalancing
2013-12-02 00:00:00 strategy [INFO] Best for class US Stocks: VTI
2013-12-02 00:00:00 strategy [INFO] Best for class Commodities: None
2013-12-02 00:00:00 strategy [INFO] Best for class US 10 Year Government Bonds: None
2013-12-02 00:00:00 strategy [INFO] Best for class Foreign Stocks: VEU
2013-12-02 00:00:00 strategy [INFO] Best for class Real Estate: None
2013-12-02 00:00:00 strategy [INFO] Placing market order for -1 VTI shares
2013-12-02 00:00:00 strategy [INFO] Placing market order for -39 VNQ shares
Sharpe ratio: -0.06
Returns: 32.97 %
```

This is what the plot should look like:



9.2 Mean Reversion

9.2.1 Ernie Chan's Gold vs. Gold Miners

This example is based on:

- http://epchan.blogspot.com.ar/2006/11/gold-vs-gold-miners-another-arbitrage.html
- https://www.quantopian.com/posts/ernie-chans-gold-vs-gold-miners-stat-arb

```
from __future__ import print_function

from quantworks import strategy
from quantworks import dataseries
from quantworks.dataseries import aligned
from quantworks import plotter
from quantworks.barfeed import yahoofeed
from quantworks.stratanalyzer import sharpe

import numpy as np
import statsmodels.api as sm


def get_beta(values1, values2):
    # http://statsmodels.sourceforge.net/stable/regression.html
    model = sm.OLS(values1, values2)
    results = model.fit()
    return results.params[0]
```

(continues on next page)

(continued from previous page)

```
class StatArbHelper:
    def __init__(self, ds1, ds2, windowSize):
        # We're going to use datetime aligned versions of the dataseries.
        self.__ds1, self.__ds2 = aligned.datetime_aligned(ds1, ds2)
        self.__windowSize = windowSize
        self.__hedgeRatio = None
        self.__spread = None
        self.__spreadMean = None
        self.__spreadStd = None
        self.__zScore = None

    def getSpread(self):
        return self.__spread

    def getSpreadMean(self):
        return self.__spreadMean

    def getSpreadStd(self):
        return self.__spreadStd

    def getZScore(self):
        return self.__zScore

    def getHedgeRatio(self):
        return self.__hedgeRatio

    def __updateHedgeRatio(self, values1, values2):
        self.__hedgeRatio = get_beta(values1, values2)

    def __updateSpreadMeanAndStd(self, values1, values2):
        if self.__hedgeRatio is not None:
            spread = values1 - values2 * self.__hedgeRatio
            self.__spreadMean = spread.mean()
            self.__spreadStd = spread.std(ddof=1)

    def __updateSpread(self):
        if self.__hedgeRatio is not None:
            self.__spread = self.__ds1[-1] - self.__hedgeRatio * self.__ds2[-1]

    def __updateZScore(self):
        if self.__spread is not None and self.__spreadMean is not None and self.__
↪spreadStd is not None:
            self.__zScore = (self.__spread - self.__spreadMean) / float(self.__
↪spreadStd)

    def update(self):
        if len(self.__ds1) >= self.__windowSize:
            values1 = np.asarray(self.__ds1[-1*self.__windowSize:])
            values2 = np.asarray(self.__ds2[-1*self.__windowSize:])
            self.__updateHedgeRatio(values1, values2)
            self.__updateSpread()
            self.__updateSpreadMeanAndStd(values1, values2)
            self.__updateZScore()


class StatArb(strategy.BacktestingStrategy):
```

(continues on next page)

(continued from previous page)

```
    def __init__(self, feed, instrument1, instrument2, windowSize):
        super(StatArb, self).__init__(feed)
        self.setUseAdjustedValues(True)
        self.__statArbHelper = StatArbHelper(feed[instrument1].
↪getAdjCloseDataSeries(), feed[instrument2].getAdjCloseDataSeries(), windowSize)
        self.__i1 = instrument1
        self.__i2 = instrument2

        # These are used only for plotting purposes.
        self.__spread = dataseries.SequenceDataSeries()
        self.__hedgeRatio = dataseries.SequenceDataSeries()

    def getSpreadDS(self):
        return self.__spread

    def getHedgeRatioDS(self):
        return self.__hedgeRatio

    def __getOrderSize(self, bars, hedgeRatio):
        cash = self.getBroker().getCash(False)
        price1 = bars[self.__i1].getAdjClose()
        price2 = bars[self.__i2].getAdjClose()
        size1 = int(cash / (price1 + hedgeRatio * price2))
        size2 = int(size1 * hedgeRatio)
        return (size1, size2)

    def buySpread(self, bars, hedgeRatio):
        amount1, amount2 = self.__getOrderSize(bars, hedgeRatio)
        self.marketOrder(self.__i1, amount1)
        self.marketOrder(self.__i2, amount2 * -1)

    def sellSpread(self, bars, hedgeRatio):
        amount1, amount2 = self.__getOrderSize(bars, hedgeRatio)
        self.marketOrder(self.__i1, amount1 * -1)
        self.marketOrder(self.__i2, amount2)

    def reducePosition(self, instrument):
        currentPos = self.getBroker().getShares(instrument)
        if currentPos > 0:
            self.marketOrder(instrument, currentPos * -1)
        elif currentPos < 0:
            self.marketOrder(instrument, currentPos * -1)

    def onBars(self, bars):
        self.__statArbHelper.update()

        # These is used only for plotting purposes.
        self.__spread.appendWithDateTime(bars.getDateTime(), self.__statArbHelper.
↪getSpread())
        self.__hedgeRatio.appendWithDateTime(bars.getDateTime(), self.__statArbHelper.
↪getHedgeRatio())

        if bars.getBar(self.__i1) and bars.getBar(self.__i2):
            hedgeRatio = self.__statArbHelper.getHedgeRatio()
            zScore = self.__statArbHelper.getZScore()
            if zScore is not None:
                currentPos = abs(self.getBroker().getShares(self.__i1)) + abs(self.
↪getBroker().getShares(self.__i2))
```

(continues on next page)

(continued from previous page)

```
                if abs(zScore) <= 1 and currentPos != 0:
                    self.reducePosition(self.__i1)
                    self.reducePosition(self.__i2)
                elif zScore <= -2 and currentPos == 0:  # Buy spread when its value␣
↪drops below 2 standard deviations.
                    self.buySpread(bars, hedgeRatio)
                elif zScore >= 2 and currentPos == 0:  # Short spread when its value␣
↪rises above 2 standard deviations.
                    self.sellSpread(bars, hedgeRatio)


def main(plot):
    instruments = ["gld", "gdx"]
    windowSize = 50

    # Load the bars. These files were manually downloaded from Yahoo Finance.
    feed = yahoofeed.Feed()
    for year in range(2006, 2012+1):
        for instrument in instruments:
            fileName = "%s-%d-yahoofinance.csv" % (instrument, year)
            print("Loading bars from %s" % fileName)
            feed.addBarsFromCSV(instrument, fileName)

    strat = StatArb(feed, instruments[0], instruments[1], windowSize)
    sharpeRatioAnalyzer = sharpe.SharpeRatio()
    strat.attachAnalyzer(sharpeRatioAnalyzer)

    if plot:
        plt = plotter.StrategyPlotter(strat, False, False, True)
        plt.getOrCreateSubplot("hedge").addDataSeries("Hedge Ratio", strat.
↪getHedgeRatioDS())
        plt.getOrCreateSubplot("spread").addDataSeries("Spread", strat.getSpreadDS())

    strat.run()
    print("Sharpe ratio: %.2f" % sharpeRatioAnalyzer.getSharpeRatio(0.05))

    if plot:
        plt.plot()


if __name__ == "__main__":
    main(True)
```

this is what the output should look like:

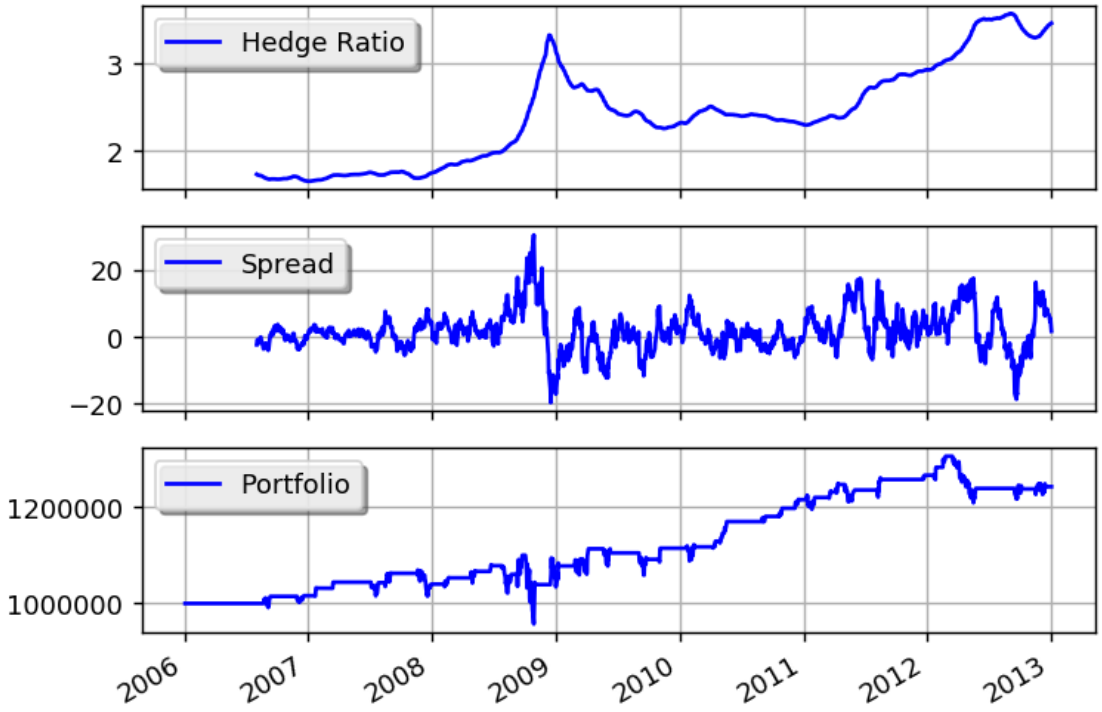
```
Loading bars from gld-2006-yahoofinance.csv
Loading bars from gdx-2006-yahoofinance.csv
Loading bars from gld-2007-yahoofinance.csv
Loading bars from gdx-2007-yahoofinance.csv
Loading bars from gld-2008-yahoofinance.csv
Loading bars from gdx-2008-yahoofinance.csv
Loading bars from gld-2009-yahoofinance.csv
Loading bars from gdx-2009-yahoofinance.csv
Loading bars from gld-2010-yahoofinance.csv
Loading bars from gdx-2010-yahoofinance.csv
Loading bars from gld-2011-yahoofinance.csv
Loading bars from gdx-2011-yahoofinance.csv
```

(continues on next page)

(continued from previous page)

```
Loading bars from gld-2012-yahoofinance.csv
Loading bars from gdx-2012-yahoofinance.csv
Sharpe ratio: -0.20
```

and this is what the plot should look like:



You can get better returns by tunning the window size as well as the entry and exit values for the z-score.

9.2.2 Bollinger Bands

This example is based on:

- http://www.investopedia.com/articles/trading/07/bollinger.asp

```
from __future__ import print_function

from quantworks import strategy
from quantworks import plotter
from quantworks.tools import quandl
from quantworks.technical import bollinger
from quantworks.stratanalyzer import sharpe
from quantworks import broker as basebroker


class BBands(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument, bBandsPeriod):
        super(BBands, self).__init__(feed)
        self.__instrument = instrument
        self.__bbands = bollinger.BollingerBands(feed[instrument].
↪getCloseDataSeries(), bBandsPeriod, 2)
```

(continues on next page)

(continued from previous page)

```
    def getBollingerBands(self):
        return self.__bbands

    def onOrderUpdated(self, order):
        if order.isBuy():
            orderType = "Buy"
        else:
            orderType = "Sell"
        self.info("%s order %d updated - Status: %s" % (
            orderType, order.getId(), basebroker.Order.State.toString(order.
↪getState())
        ))

    def onBars(self, bars):
        lower = self.__bbands.getLowerBand()[-1]
        upper = self.__bbands.getUpperBand()[-1]
        if lower is None:
            return

        shares = self.getBroker().getShares(self.__instrument)
        bar = bars[self.__instrument]
        if shares == 0 and bar.getClose() < lower:
            sharesToBuy = int(self.getBroker().getCash(False) / bar.getClose())
            self.info("Placing buy market order for %s shares" % sharesToBuy)
            self.marketOrder(self.__instrument, sharesToBuy)
        elif shares > 0 and bar.getClose() > upper:
            self.info("Placing sell market order for %s shares" % shares)
            self.marketOrder(self.__instrument, -1*shares)


def main(plot):
    instrument = "yhoo"
    bBandsPeriod = 40

    # Download the bars.
    feed = quandl.build_feed("WIKI", [instrument], 2011, 2012, ".")

    strat = BBands(feed, instrument, bBandsPeriod)
    sharpeRatioAnalyzer = sharpe.SharpeRatio()
    strat.attachAnalyzer(sharpeRatioAnalyzer)

    if plot:
        plt = plotter.StrategyPlotter(strat, True, True, True)
        plt.getInstrumentSubplot(instrument).addDataSeries("upper", strat.
↪getBollingerBands().getUpperBand())
        plt.getInstrumentSubplot(instrument).addDataSeries("middle", strat.
↪getBollingerBands().getMiddleBand())
        plt.getInstrumentSubplot(instrument).addDataSeries("lower", strat.
↪getBollingerBands().getLowerBand())

    strat.run()
    print("Sharpe ratio: %.2f" % sharpeRatioAnalyzer.getSharpeRatio(0.05))

    if plot:
        plt.plot()
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    main(True)
```

this is what the output should look like:

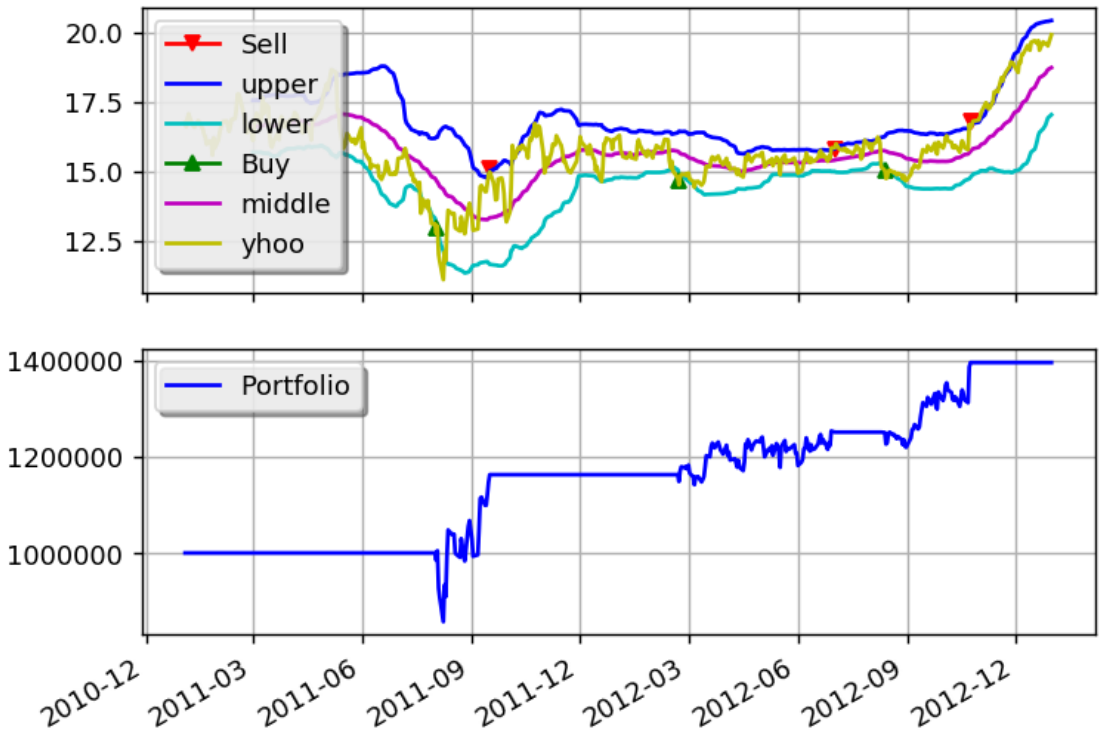
```
2011-07-20 00:00:00 strategy [INFO] Placing buy market order for 74183 shares
2011-07-20 00:00:00 strategy [INFO] Buy order 1 updated - Status: SUBMITTED
2011-07-21 00:00:00 strategy [INFO] Buy order 1 updated - Status: ACCEPTED
2011-07-21 00:00:00 broker.backtesting [DEBUG] Not enough cash to fill yhoo order [1]␣
↪for 74183 share/s
2011-07-21 00:00:00 strategy [INFO] Buy order 1 updated - Status: CANCELED
2011-07-21 00:00:00 strategy [INFO] Placing buy market order for 73583 shares
2011-07-21 00:00:00 strategy [INFO] Buy order 2 updated - Status: SUBMITTED
2011-07-22 00:00:00 strategy [INFO] Buy order 2 updated - Status: ACCEPTED
2011-07-22 00:00:00 broker.backtesting [DEBUG] Not enough cash to fill yhoo order [2]␣
↪for 73583 share/s
2011-07-22 00:00:00 strategy [INFO] Buy order 2 updated - Status: CANCELED
2011-07-25 00:00:00 strategy [INFO] Placing buy market order for 73046 shares
2011-07-25 00:00:00 strategy [INFO] Buy order 3 updated - Status: SUBMITTED
2011-07-26 00:00:00 strategy [INFO] Buy order 3 updated - Status: ACCEPTED
2011-07-26 00:00:00 broker.backtesting [DEBUG] Not enough cash to fill yhoo order [3]␣
↪for 73046 share/s
2011-07-26 00:00:00 strategy [INFO] Buy order 3 updated - Status: CANCELED
2011-07-27 00:00:00 strategy [INFO] Placing buy market order for 73610 shares
2011-07-27 00:00:00 strategy [INFO] Buy order 4 updated - Status: SUBMITTED
2011-07-28 00:00:00 strategy [INFO] Buy order 4 updated - Status: ACCEPTED
2011-07-28 00:00:00 broker.backtesting [DEBUG] Not enough cash to fill yhoo order [4]␣
↪for 73610 share/s
2011-07-28 00:00:00 strategy [INFO] Buy order 4 updated - Status: CANCELED
2011-07-28 00:00:00 strategy [INFO] Placing buy market order for 74074 shares
2011-07-28 00:00:00 strategy [INFO] Buy order 5 updated - Status: SUBMITTED
2011-07-29 00:00:00 strategy [INFO] Buy order 5 updated - Status: ACCEPTED
2011-07-29 00:00:00 broker.backtesting [DEBUG] Not enough cash to fill yhoo order [5]␣
↪for 74074 share/s
2011-07-29 00:00:00 strategy [INFO] Buy order 5 updated - Status: CANCELED
2011-07-29 00:00:00 strategy [INFO] Placing buy market order for 76335 shares
2011-07-29 00:00:00 strategy [INFO] Buy order 6 updated - Status: SUBMITTED
2011-08-01 00:00:00 strategy [INFO] Buy order 6 updated - Status: ACCEPTED
2011-08-01 00:00:00 broker.backtesting [DEBUG] Not enough cash to fill yhoo order [6]␣
↪for 76335 share/s
2011-08-01 00:00:00 strategy [INFO] Buy order 6 updated - Status: CANCELED
2011-08-01 00:00:00 strategy [INFO] Placing buy market order for 76335 shares
2011-08-01 00:00:00 strategy [INFO] Buy order 7 updated - Status: SUBMITTED
2011-08-02 00:00:00 strategy [INFO] Buy order 7 updated - Status: ACCEPTED
2011-08-02 00:00:00 strategy [INFO] Buy order 7 updated - Status: FILLED
2011-09-15 00:00:00 strategy [INFO] Placing sell market order for 76335 shares
2011-09-15 00:00:00 strategy [INFO] Sell order 8 updated - Status: SUBMITTED
2011-09-16 00:00:00 strategy [INFO] Sell order 8 updated - Status: ACCEPTED
2011-09-16 00:00:00 strategy [INFO] Sell order 8 updated - Status: FILLED
2012-02-17 00:00:00 strategy [INFO] Placing buy market order for 77454 shares
2012-02-17 00:00:00 strategy [INFO] Buy order 9 updated - Status: SUBMITTED
2012-02-21 00:00:00 strategy [INFO] Buy order 9 updated - Status: ACCEPTED
2012-02-21 00:00:00 broker.backtesting [DEBUG] Not enough cash to fill yhoo order [9]␣
↪for 77454 share/s
2012-02-21 00:00:00 strategy [INFO] Buy order 9 updated - Status: CANCELED
2012-02-21 00:00:00 strategy [INFO] Placing buy market order for 78819 shares
```

(continues on next page)

(continued from previous page)

```
2012-02-21 00:00:00 strategy [INFO] Buy order 10 updated - Status: SUBMITTED
2012-02-22 00:00:00 strategy [INFO] Buy order 10 updated - Status: ACCEPTED
2012-02-22 00:00:00 strategy [INFO] Buy order 10 updated - Status: FILLED
2012-06-29 00:00:00 strategy [INFO] Placing sell market order for 78819 shares
2012-06-29 00:00:00 strategy [INFO] Sell order 11 updated - Status: SUBMITTED
2012-07-02 00:00:00 strategy [INFO] Sell order 11 updated - Status: ACCEPTED
2012-07-02 00:00:00 strategy [INFO] Sell order 11 updated - Status: FILLED
2012-08-10 00:00:00 strategy [INFO] Placing buy market order for 82565 shares
2012-08-10 00:00:00 strategy [INFO] Buy order 12 updated - Status: SUBMITTED
2012-08-13 00:00:00 strategy [INFO] Buy order 12 updated - Status: ACCEPTED
2012-08-13 00:00:00 strategy [INFO] Buy order 12 updated - Status: FILLED
2012-10-23 00:00:00 strategy [INFO] Placing sell market order for 82565 shares
2012-10-23 00:00:00 strategy [INFO] Sell order 13 updated - Status: SUBMITTED
2012-10-24 00:00:00 strategy [INFO] Sell order 13 updated - Status: ACCEPTED
2012-10-24 00:00:00 strategy [INFO] Sell order 13 updated - Status: FILLED
Sharpe ratio: 0.71
```

and this is what the plot should look like:



You can get better returns by tunning the Bollinger Bands period as well as the entry and exit points.

9.2.3 RSI2

This example is based on a strategy known as RSI2 (http://stockcharts.com/school/doku.php?id=chart_school:trading_strategies:rsi2) which requires the following parameters:

- An SMA period for trend identification. We'll call this entrySMA.
- A smaller SMA period for the exit point. We'll call this exitSMA.

- An RSI period for entering both short/long positions. We'll call this rsiPeriod.
- An RSI oversold threshold for long position entry. We'll call this overSoldThreshold.
- An RSI overbought threshold for short position entry. We'll call this overBoughtThreshold.

Save this code as rsi2.py:

```
from quantworks import strategy
from quantworks.technical import ma
from quantworks.technical import rsi
from quantworks.technical import cross


class RSI2(strategy.BacktestingStrategy):
    def __init__(self, feed, instrument, entrySMA, exitSMA, rsiPeriod,␣
↪overBoughtThreshold, overSoldThreshold):
        super(RSI2, self).__init__(feed)
        self.__instrument = instrument
        # We'll use adjusted close values, if available, instead of regular close␣
↪values.
        if feed.barsHaveAdjClose():
            self.setUseAdjustedValues(True)
        self.__priceDS = feed[instrument].getPriceDataSeries()
        self.__entrySMA = ma.SMA(self.__priceDS, entrySMA)
        self.__exitSMA = ma.SMA(self.__priceDS, exitSMA)
        self.__rsi = rsi.RSI(self.__priceDS, rsiPeriod)
        self.__overBoughtThreshold = overBoughtThreshold
        self.__overSoldThreshold = overSoldThreshold
        self.__longPos = None
        self.__shortPos = None

    def getEntrySMA(self):
        return self.__entrySMA

    def getExitSMA(self):
        return self.__exitSMA

    def getRSI(self):
        return self.__rsi

    def onEnterCanceled(self, position):
        if self.__longPos == position:
            self.__longPos = None
        elif self.__shortPos == position:
            self.__shortPos = None
        else:
            assert(False)

    def onExitOk(self, position):
        if self.__longPos == position:
            self.__longPos = None
        elif self.__shortPos == position:
            self.__shortPos = None
        else:
            assert(False)

    def onExitCanceled(self, position):
        # If the exit was canceled, re-submit it.
```

(continues on next page)

(continued from previous page)

```
        position.exitMarket()

    def onBars(self, bars):
        # Wait for enough bars to be available to calculate SMA and RSI.
        if self.__exitSMA[-1] is None or self.__entrySMA[-1] is None or self.__rsi[-
↪1] is None:
            return

        bar = bars[self.__instrument]
        if self.__longPos is not None:
            if self.exitLongSignal():
                self.__longPos.exitMarket()
        elif self.__shortPos is not None:
            if self.exitShortSignal():
                self.__shortPos.exitMarket()
        else:
            if self.enterLongSignal(bar):
                shares = int(self.getBroker().getCash() * 0.9 / bars[self.__
↪instrument].getPrice())
                self.__longPos = self.enterLong(self.__instrument, shares, True)
            elif self.enterShortSignal(bar):
                shares = int(self.getBroker().getCash() * 0.9 / bars[self.__
↪instrument].getPrice())
                self.__shortPos = self.enterShort(self.__instrument, shares, True)

    def enterLongSignal(self, bar):
        return bar.getPrice() > self.__entrySMA[-1] and self.__rsi[-1] <= self.__
↪overSoldThreshold

    def exitLongSignal(self):
        return cross.cross_above(self.__priceDS, self.__exitSMA) and not self.__
↪longPos.exitActive()

    def enterShortSignal(self, bar):
        return bar.getPrice() < self.__entrySMA[-1] and self.__rsi[-1] >= self.__
↪overBoughtThreshold

    def exitShortSignal(self):
        return cross.cross_below(self.__priceDS, self.__exitSMA) and not self.__
↪shortPos.exitActive()
```

and use the following code to execute the strategy:

```
from __future__ import print_function

import rsi2
from quantworks import plotter
from quantworks.barfeed import yahoofeed
from quantworks.stratanalyzer import sharpe


def main(plot):
    instrument = "DIA"
    entrySMA = 200
    exitSMA = 5
    rsiPeriod = 2
    overBoughtThreshold = 90
```

(continues on next page)

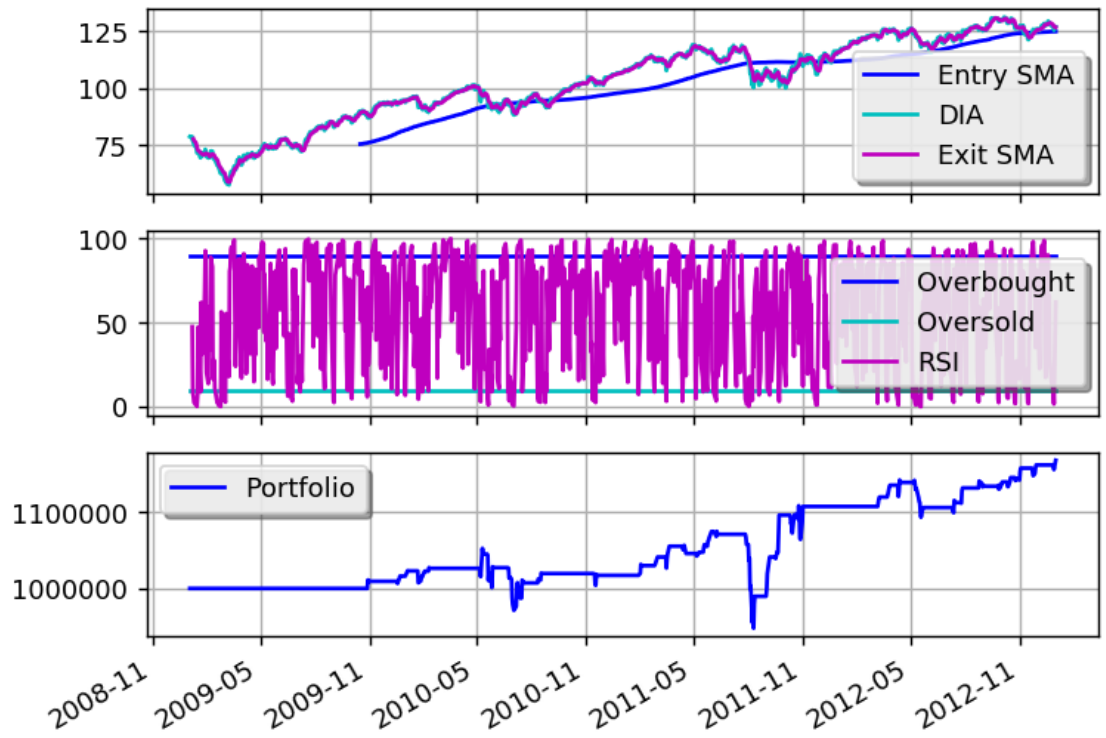
(continued from previous page)

```
    overSoldThreshold = 10

    # Load the bars. These files were manually downloaded from Yahoo Finance.
    feed = yahoofeed.Feed()
    for year in range(2009, 2013):
        fileName = "%s-%d-yahoofinance.csv" % (instrument, year)
        print("Loading bars from %s" % fileName)
        feed.addBarsFromCSV(instrument, fileName)

    strat = rsi2.RSI2(feed, instrument, entrySMA, exitSMA, rsiPeriod,␣
↪overBoughtThreshold, overSoldThreshold)
    sharpeRatioAnalyzer = sharpe.SharpeRatio()
    strat.attachAnalyzer(sharpeRatioAnalyzer)

    if plot:
        plt = plotter.StrategyPlotter(strat, True, False, True)
        plt.getInstrumentSubplot(instrument).addDataSeries("Entry SMA", strat.
↪getEntrySMA())
        plt.getInstrumentSubplot(instrument).addDataSeries("Exit SMA", strat.
↪getExitSMA())
        plt.getOrCreateSubplot("rsi").addDataSeries("RSI", strat.getRSI())
        plt.getOrCreateSubplot("rsi").addLine("Overbought", overBoughtThreshold)
        plt.getOrCreateSubplot("rsi").addLine("Oversold", overSoldThreshold)

    strat.run()
    print("Sharpe ratio: %.2f" % sharpeRatioAnalyzer.getSharpeRatio(0.05))

    if plot:
        plt.plot()


if __name__ == "__main__":
    main(True)
```

This is what the output should look like:

```
Loading bars from DIA-2009-yahoofinance.csv
Loading bars from DIA-2010-yahoofinance.csv
Loading bars from DIA-2011-yahoofinance.csv
Loading bars from DIA-2012-yahoofinance.csv
Sharpe ratio: -0.11
```

and this is what the plot should look like:



You can get better returns by tunning the different parameters.

9.3 Others

9.3.1 Quandl integration

The purpose of this example is to show how to integrate price data along with any time-series data in CSV format from Quandl into a strategy.

We'll use the following CSV data from Quandl: http://www.quandl.com/OFDP-Open-Financial-Data-Project/GOLD_2-LBMA-Gold-Price-London-Fixings-P-M

```
from quantworks import strategy
from quantworks import plotter
from quantworks.tools import quandl
from quantworks.feed import csvfeed
import datetime


class MyStrategy(strategy.BacktestingStrategy):
    def __init__(self, feed, quandlFeed, instrument):
        super(MyStrategy, self).__init__(feed)
        self.setUseAdjustedValues(True)
        self.__instrument = instrument

        # It is VERY important to add the the extra feed to the event dispatch loop␣
↪before
        # running the strategy.
        self.getDispatcher().addSubject(quandlFeed)
```

(continues on next page)

(continued from previous page)

```
        # Subscribe to events from the Quandl feed.
        quandlFeed.getNewValuesEvent().subscribe(self.onQuandlData)

    def onQuandlData(self, dateTime, values):
        self.info(values)

    def onBars(self, bars):
        self.info(bars[self.__instrument].getAdjClose())


def main(plot):
    instruments = ["GORO"]

    # Download GORO bars using WIKI source code.
    feed = quandl.build_feed("WIKI", instruments, 2006, 2012, ".")

    # Load Quandl CSV downloaded from http://www.quandl.com/OFDP-Open-Financial-Data-
↪Project/GOLD_2-LBMA-Gold-Price-London-Fixings-P-M
    quandlFeed = csvfeed.Feed("Date", "%Y-%m-%d")
    quandlFeed.setDateRange(datetime.datetime(2006, 1, 1), datetime.datetime(2012, 12,
↪ 31))
    quandlFeed.addValuesFromCSV("quandl_gold_2.csv")

    myStrategy = MyStrategy(feed, quandlFeed, instruments[0])

    if plot:
        plt = plotter.StrategyPlotter(myStrategy, True, False, False)
        plt.getOrCreateSubplot("quandl").addDataSeries("USD", quandlFeed["USD"])
        plt.getOrCreateSubplot("quandl").addDataSeries("EUR", quandlFeed["EUR"])
        plt.getOrCreateSubplot("quandl").addDataSeries("GBP", quandlFeed["GBP"])

    myStrategy.run()

    if plot:
        plt.plot()


if __name__ == "__main__":
    main(True)
```

This is what the output should look like:

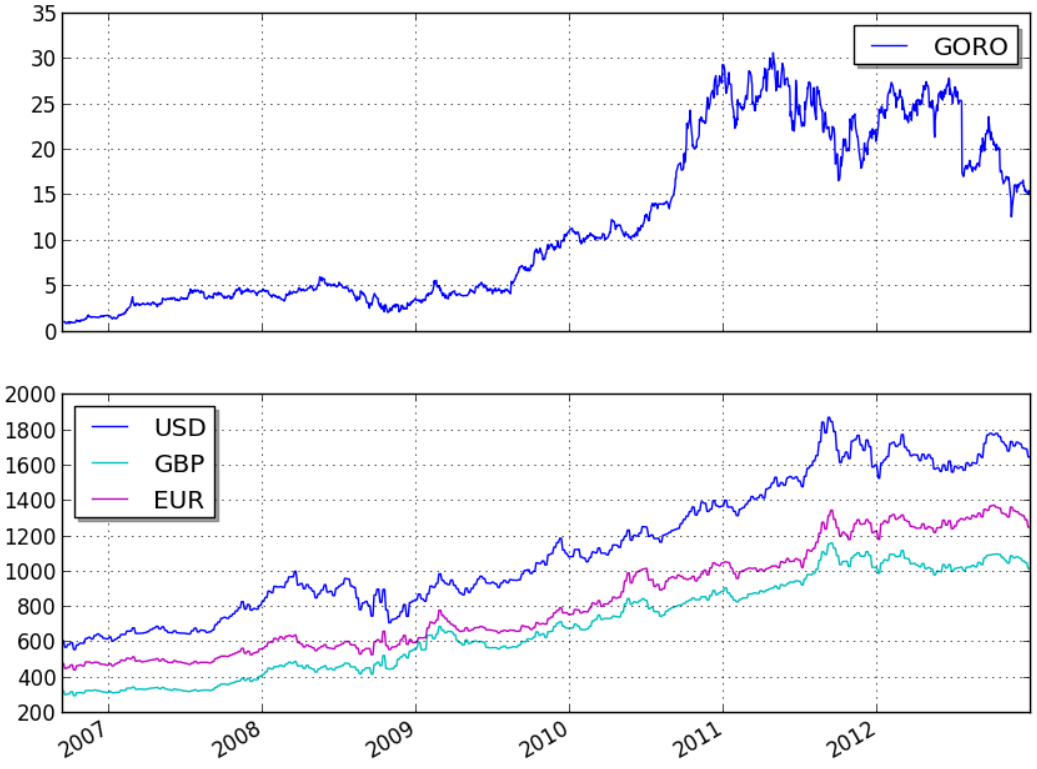
```
2006-01-01 00:00:00 strategy [INFO] {'USD': 513.0, 'GBP': 298.204, 'EUR': 433.533}
2006-01-08 00:00:00 strategy [INFO] {'USD': 535.25, 'GBP': 302.572, 'EUR': 440.173}
2006-01-15 00:00:00 strategy [INFO] {'USD': 548.25, 'GBP': 309.781, 'EUR': 454.489}
2006-01-22 00:00:00 strategy [INFO] {'USD': 567.25, 'GBP': 321.152, 'EUR': 468.802}
2006-01-29 00:00:00 strategy [INFO] {'USD': 561.75, 'GBP': 315.147, 'EUR': 460.526}
2006-02-05 00:00:00 strategy [INFO] {'USD': 569.0, 'GBP': 322.562, 'EUR': 474.167}
2006-02-12 00:00:00 strategy [INFO] {'USD': 557.0, 'GBP': 317.198, 'EUR': 463.78}
2006-02-19 00:00:00 strategy [INFO] {'USD': 551.7, 'GBP': 317.251, 'EUR': 463.224}
2006-02-26 00:00:00 strategy [INFO] {'USD': 554.15, 'GBP': 316.838, 'EUR': 465.555}
2006-03-05 00:00:00 strategy [INFO] {'USD': 565.0, 'GBP': 322.029, 'EUR': 469.854}
.
.
.
2012-12-19 00:00:00 strategy [INFO] 15.43
```

(continues on next page)

(continued from previous page)

```
2012-12-20 00:00:00 strategy [INFO] 15.39
2012-12-21 00:00:00 strategy [INFO] 15.35
2012-12-23 00:00:00 strategy [INFO] {'USD': 1651.5, 'GBP': 1019.256, 'EUR': 1253.701}
2012-12-24 00:00:00 strategy [INFO] 15.2
2012-12-26 00:00:00 strategy [INFO] 15.56
2012-12-27 00:00:00 strategy [INFO] 15.24
2012-12-28 00:00:00 strategy [INFO] 15.09
2012-12-30 00:00:00 strategy [INFO] {'USD': 1657.5, 'GBP': 1027.206, 'EUR': 1253.024}
2012-12-31 00:00:00 strategy [INFO] 15.41
```

and this is what the plot should look like:



CHAPTER 10

Contributing to QuantWorks

10.1 Reporting Issues

- Please check the GitHub issue tracker
- Please add an issue with specific requirements
- Please include a traceback/exception/error message and reproducible use case if you report a bug

10.2 Contributing Code

- Please fork the repo and add your code to a feature branch, and open a PR against this repo
- **You will notice this repo follows a non-standard style convention. We respect the original PyAlgoTrade conventions**
  - camelCaseVariables and methodNames
  - liberal use of __fullyPrivateVariables and getPrivateVariable() and setPrivateVariable()
  - prolific use of abstract class interfaces, base classes and implementation classes that hide their base

If you have any questions on style conventions please open an issue! We are happy to discuss and begin building a style guide document.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

q

quantworks.bar, 21
quantworks.broker, 26
quantworks.broker.backtesting, 31
quantworks.broker.fillstrategy, 36
quantworks.broker.slippage, 35
quantworks.marketsession, 44
quantworks.optimizer.server, 44
quantworks.stratanalyzer, 40
quantworks.stratanalyzer.drawdown, 40

Index

Symbols

__contains__() (*quantworks.bar.Bars method*), 23
__getitem__() (*quantworks.bar.Bars method*), 23
__init__() (*quantworks.bar.Bars method*), 23
__init__() (*quantworks.bar.BasicBar method*), 22

B

Bar (*class in quantworks.bar*), 21
Bars (*class in quantworks.bar*), 22
BasicBar (*class in quantworks.bar*), 22
BOVESPA (*class in quantworks.marketsession*), 45
Broker (*class in quantworks.broker*), 29
Broker (*class in quantworks.broker.backtesting*), 32

C

calculate() (*quantworks.broker.backtesting.Commission method*), 31
calculate() (*quantworks.broker.backtesting.FixedPerTrade method*), 32
calculate() (*quantworks.broker.backtesting.NoCommission method*), 32
calculate() (*quantworks.broker.backtesting.TradePercentage method*), 32
calculatePrice() (*quantworks.broker.slippage.NoSlippage method*), 35
calculatePrice() (*quantworks.broker.slippage.SlippageModel method*), 35
calculatePrice() (*quantworks.broker.slippage.VolumeShareSlippage method*), 36
cancelOrder() (*quantworks.broker.backtesting.Broker method*), 33
cancelOrder() (*quantworks.broker.Broker method*), 31
Commission (*class in quantworks.broker.backtesting*), 31
createLimitOrder() (*quantworks.broker.backtesting.Broker method*), 33
createLimitOrder() (*quantworks.broker.Broker method*), 30
createMarketOrder() (*quantworks.broker.backtesting.Broker method*), 33
createMarketOrder() (*quantworks.broker.Broker method*), 30
createStopLimitOrder() (*quantworks.broker.backtesting.Broker method*), 33
createStopLimitOrder() (*quantworks.broker.Broker method*), 31
createStopOrder() (*quantworks.broker.backtesting.Broker method*), 34
createStopOrder() (*quantworks.broker.Broker method*), 30

D

DefaultStrategy (*class in quantworks.broker.fillstrategy*), 37
dispatch() (*quantworks.broker.backtesting.Broker method*), 34
DrawDown (*class in quantworks.stratanalyzer.drawdown*), 40

E

eof() (*quantworks.broker.backtesting.Broker method*), 34

F

fillLimitOrder() (*quantworks.broker.fillstrategy.DefaultStrategy*

method), 38
fillLimitOrder() (*quantworks.broker.fillstrategy.FillStrategy method*), 36
fillMarketOrder() (*quantworks.broker.fillstrategy.DefaultStrategy method*), 38
fillMarketOrder() (*quantworks.broker.fillstrategy.FillStrategy method*), 36
fillStopLimitOrder() (*quantworks.broker.fillstrategy.DefaultStrategy method*), 38
fillStopLimitOrder() (*quantworks.broker.fillstrategy.FillStrategy method*), 37
fillStopOrder() (*quantworks.broker.fillstrategy.DefaultStrategy method*), 39
fillStopOrder() (*quantworks.broker.fillstrategy.FillStrategy method*), 37
FillStrategy (*class in quantworks.broker.fillstrategy*), 36
FixedPerTrade (*class in quantworks.broker.backtesting*), 32
Frequency (*class in quantworks.bar*), 21
FTSE (*class in quantworks.marketsession*), 45

G

getAction() (*quantworks.broker.Order method*), 27
getActiveOrders() (*quantworks.broker.backtesting.Broker method*), 34
getActiveOrders() (*quantworks.broker.Broker method*), 30
getAdjClose() (*quantworks.bar.Bar method*), 22
getAdjClose() (*quantworks.bar.BasicBar method*), 22
getAllOrNone() (*quantworks.broker.Order method*), 28
getAvgFillPrice() (*quantworks.broker.Order method*), 28
getBar() (*quantworks.bar.Bars method*), 23
getCash() (*quantworks.broker.backtesting.Broker method*), 34
getCash() (*quantworks.broker.Broker method*), 29
getClose() (*quantworks.bar.Bar method*), 22
getClose() (*quantworks.bar.BasicBar method*), 22
getCommission() (*quantworks.broker.backtesting.Broker method*), 34
getCommission() (*quantworks.broker.OrderExecutionInfo method*), 29
getDateTime() (*quantworks.bar.Bar method*), 21
getDateTime() (*quantworks.bar.Bars method*), 23
getDateTime() (*quantworks.bar.BasicBar method*), 22
getDateTime() (*quantworks.broker.OrderExecutionInfo method*), 29
getEquity() (*quantworks.broker.backtesting.Broker method*), 34
getExecutionInfo() (*quantworks.broker.Order method*), 28
getFilled() (*quantworks.broker.Order method*), 28
getFillOnClose() (*quantworks.broker.MarketOrder method*), 28
getFillStrategy() (*quantworks.broker.backtesting.Broker method*), 34
getFrequency() (*quantworks.bar.Bar method*), 22
getFrequency() (*quantworks.bar.BasicBar method*), 22
getGoodTillCanceled() (*quantworks.broker.Order method*), 28
getHigh() (*quantworks.bar.Bar method*), 21
getHigh() (*quantworks.bar.BasicBar method*), 22
getId() (*quantworks.broker.Order method*), 26
getInstrument() (*quantworks.broker.Order method*), 27
getInstruments() (*quantworks.bar.Bars method*), 23
getLimitPrice() (*quantworks.broker.LimitOrder method*), 28
getLimitPrice() (*quantworks.broker.StopLimitOrder method*), 29
getLongestDrawDownDuration() (*quantworks.stratanalyzer.drawdown.DrawDown method*), 40
getLow() (*quantworks.bar.Bar method*), 22
getLow() (*quantworks.bar.BasicBar method*), 22
getMaxDrawDown() (*quantworks.stratanalyzer.drawdown.DrawDown method*), 40
getOpen() (*quantworks.bar.Bar method*), 21
getOpen() (*quantworks.bar.BasicBar method*), 22
getParameters() (*quantworks.optimizer.server.Results method*), 44
getPositions() (*quantworks.broker.backtesting.Broker method*), 34
getPositions() (*quantworks.broker.Broker method*), 29
getPrice() (*quantworks.bar.Bar method*), 22
getPrice() (*quantworks.bar.BasicBar method*), 22

getPrice() (*quantworks.broker.OrderExecutionInfo method*), 29
getQuantity() (*quantworks.broker.Order method*), 27
getQuantity() (*quantworks.broker.OrderExecutionInfo method*), 29
getRemaining() (*quantworks.broker.Order method*), 28
getResult() (*quantworks.optimizer.server.Results method*), 44
getShares() (*quantworks.broker.backtesting.Broker method*), 34
getShares() (*quantworks.broker.Broker method*), 29
getState() (*quantworks.broker.Order method*), 27
getStopPrice() (*quantworks.broker.StopLimitOrder method*), 29
getStopPrice() (*quantworks.broker.StopOrder method*), 29
getSubmitDateTime() (*quantworks.broker.Order method*), 27
getTimezone() (*quantworks.marketsession.MarketSession class method*), 44
getType() (*quantworks.broker.Order method*), 27
getTypicalPrice() (*quantworks.bar.Bar method*), 22
getVolume() (*quantworks.bar.Bar method*), 22
getVolume() (*quantworks.bar.BasicBar method*), 22

I

isAccepted() (*quantworks.broker.Order method*), 27
isActive() (*quantworks.broker.Order method*), 27
isCanceled() (*quantworks.broker.Order method*), 27
isFilled() (*quantworks.broker.Order method*), 27
isInitial() (*quantworks.broker.Order method*), 27
isPartiallyFilled() (*quantworks.broker.Order method*), 27
isSubmitted() (*quantworks.broker.Order method*), 27

L

LimitOrder (*class in quantworks.broker*), 28

M

MarketOrder (*class in quantworks.broker*), 28
MarketSession (*class in quantworks.marketsession*), 44
MERVAL (*class in quantworks.marketsession*), 45

N

NASDAQ (*class in quantworks.marketsession*), 44
NoCommission (*class in quantworks.broker.backtesting*), 31
NoSlippage (*class in quantworks.broker.slippage*), 35
NYSE (*class in quantworks.marketsession*), 45

O

onBars() (*quantworks.broker.fillstrategy.DefaultStrategy method*), 39
onBars() (*quantworks.broker.fillstrategy.FillStrategy method*), 37
onOrderFilled() (*quantworks.broker.fillstrategy.DefaultStrategy method*), 39
onOrderFilled() (*quantworks.broker.fillstrategy.FillStrategy method*), 37
Order (*class in quantworks.broker*), 26
OrderExecutionInfo (*class in quantworks.broker*), 29

P

peekDateTime() (*quantworks.broker.backtesting.Broker method*), 34

Q

quantworks.bar (*module*), 21
quantworks.broker (*module*), 26
quantworks.broker.backtesting (*module*), 31
quantworks.broker.fillstrategy (*module*), 36
quantworks.broker.slippage (*module*), 35
quantworks.marketsession (*module*), 44
quantworks.optimizer.server (*module*), 44
quantworks.stratanalyzer (*module*), 40
quantworks.stratanalyzer.drawdown (*module*), 40

R

Results (*class in quantworks.optimizer.server*), 44

S

serve() (*in module quantworks.optimizer.server*), 44
setAllOrNone() (*quantworks.broker.Order method*), 28
setCommission() (*quantworks.broker.backtesting.Broker method*), 34
setFillStrategy() (*quantworks.broker.backtesting.Broker method*), 35
setGoodTillCanceled() (*quantworks.broker.Order method*), 28
setShares() (*quantworks.broker.backtesting.Broker method*), 35

setSlippageModel() (*quantworks.broker.fillstrategy.DefaultStrategy method*), 39
setVolumeLimit() (*quantworks.broker.fillstrategy.DefaultStrategy method*), 39
SlippageModel (*class in quantworks.broker.slippage*), 35
StopLimitOrder (*class in quantworks.broker*), 29
StopOrder (*class in quantworks.broker*), 29
StrategyAnalyzer (*class in quantworks.stratanalyzer*), 40
submitOrder() (*quantworks.broker.backtesting.Broker method*), 35
submitOrder() (*quantworks.broker.Broker method*), 30

T

TradePercentage (*class in quantworks.broker.backtesting*), 32
TSE (*class in quantworks.marketsession*), 45

U

USEquities (*class in quantworks.marketsession*), 45

V

VolumeShareSlippage (*class in quantworks.broker.slippage*), 36